Programmierwettbewerb
Fleißige Biber
Amiga 2000 zu gewinnen

64'er

4

...e Mikrocomputer-Zeitschrift

7 DM · 60 öS · 7 sfr. · April 1987

Grafik:

CAD mit EGA

Marktübersicht:
PC-Steckkarten

Neuheiten:
CeBIT '87

Grundlagen:
Fehlerbehandlung
in Turbo-Pascal

Interruptbehandlung
am 680XX

Atari ST:
Omikron-Basic

Liebe Leser!

Kompatibilität sei das Ende des Fortschrittes, so hat mc sinngemäß früher einmal ausgerufen. Klar, denn wenn alles Norm ist, dann ist an einem Gerät nichts Neues mehr dran. In diesem Heft können Sie aber feststellen, daß insgeheim doch mehr Fortschritt stattfindet, als mancher sich träumen läßt. Ein Kenner der Kompatiblen-Szene, der selbst das Original immer wieder nach Kundenwünschen ausstatten hilft, stößt in großer Not die Frage aus, ob denn IBM noch zu sich selbst kompatibel sei! Dies ist nun der Punkt, den man sich auf der Zunge zergehen lassen muß. IBM ist viel innovativer, als alle Nachahmer, von Amdahl (Groß-Computer) bis zum Billigst-PC-Klonierer (Taiwan et al.), sich träumen lassen. IBM sorgt sich darum, daß zum Beispiel die Festplatten in seinen PCs optimal installiert werden. Welche Probleme das für den mit sich bringt, der selbst eine nachrüsten will, das können Sie in diesem Heft nachlesen. Wie gesagt, mit der Stoß-Seufzer-Frage: „wie kompatibel ... ???"

Marktübersichten scheinen beliebt bei den Lesern zu sein. Nicht anders kann erklärt werden, daß unsere Bemühungen vor Weihnachten ein so positives Echo gefunden haben. Mancher Redakteur im Kollegium hat nämlich vorher verächtlich gefragt: Was soll das denn, kleines Bildchen, knappe Beschreibung und vielleicht noch eine Fußnote über Besonderheiten? Die Antwort: Oft fehlt einem Computeristen draußen im Lande einfach die banale Information, daß es dieses Gerät XY mit den Eigenschaften YZ überhaupt gibt! Und zwar gerade dann, wenn er es brauchen könnte. Information ist genau das, was Redaktionen für die Leser aufbereiten sollen. Wir haben deshalb Informationen

über 90 PC-Einsteck-Karten gesammelt und sind stolz, deren Kurzportrait jetzt in mc liefern zu können: Wie sie aussehen, was sie machen und wer sie liefert.

Eine ganz alte Sache wurde neu aufgelegt. 1983 hat mc zusammen mit Wolfgang Back, dem Redakteur der WDR-Fernseh-Sendereihe „WDR-Computer-Club" den Know-how-Computer entwickelt. Das ist ein Papierformular mit einigem Text darum herum, das aufzeigt, wie ein Computer arbeitet. Über 100 000 dieser billigsten Computer der Welt (sie kosteten nur das Porto) sind damals vom WDR und vom Verlag verteilt worden. Da Wolfgang Back (und wir bei mc) auch heute noch immer wieder nach dem Know-how-Computer gefragt werden, haben wir dieses Objekt mit einigen Ergänzungen neu aufgelegt. Dazugekommen ist ein Text „Computer für Anfänger", der die Prinzipien noch weitergehend schildert und eine Papier-CPU, die exakt erläutert, was in einer mikroprogrammierbaren CPU so vor sich geht. Das Paket können Sie gegen 2,50 DM Porto in Briefmarken unter dem Stichwort „know-how-Computer" beim Verlag bestellen. Lehrer, die Klassensätze bestellen wollen, rufen beim Verlag unter der Nummer 0 89/51 17-3 37 an, um die Bedingungen zu erfahren.

Unsere Leserumfrage hat uns umgeworfen. Die Leser – also Sie – haben uns so positiv beurteilt, daß wir uns genieren, das in Zahlen öffentlich zu machen, es klingt zu sehr nach Selbstlob. Wir bedanken uns deshalb für das Vertrauen, das Sie in uns setzen. Das Bild zeigt unsere Redaktions-Assistentin Rita Schleser, in den Einsendungen wühlend (jetzt lacht sie wieder, aber sie hat Tausende von ausgefüllten Fragebögen aus dem Kuvert holen müssen!), um den Gewinner des Atari und des Plotters von Habersetzer zu ermitteln. Es sind Herr Elmar Weiß, 4390 Gladbeck (Atari 1040 ST) und Herr Wolfgang Kaiser, 6108 Weiterstadt (Plotter von Habersetzer).

Herzlichen Glückwunsch! Für alle Leser mit Interesse an den Grundlagen der Informatik gibt es übrigens in diesem Heft wieder etwas zu gewinnen: einen Amiga 2000 von Commodore! Die Aufgabe lautet: Konstruieren Sie den fleißigsten Biber.

Herzlichst

Ihr

Ulrich Rohde

mc-inhalt

mc-editorial

Liebe Leser 3

mc-briefe 6

mc-info 10

Spruch des Monats 48

mc-bücher 26

mc-grundlagen

Wenn die neue Harddisk nicht läuft 36
Ein Exkurs in die Technik von Festplatten und Controllern sorgt für den nötigen Durchblick bei der Installation der preiswerten Massenspeicher

Ist IBM zu sich selbst noch kompatibel 39
Nicht alle DOS-Versionen sind verträglich

Interrupts bei Prozessoren 680XX 40
Im zweiten Teil unserer Serie erfahren Sie, welche Unterschiede es bei den 680XX-Prozessoren gibt und welche Regeln bei der Programmierung von Interrupt-Routinen zu beachten sind

mc-marktübersicht

PC-Erweiterungskarten 49
Nach einer Erklärung der wichtigsten Abkürzungen, die bei Erweiterungskarten immer wieder auftauchen, informiert mc Sie über 90 PC-Erweiterungskarten, die wir Ihnen gleich mit Bild vorstellen können

mc-soft

Turbo ruft Command 60
Über den (undokumentierten) Interrupt $2E kann Turbo-Pascal auch unter DOS 2.11 Betriebssystembefehle aufrufen

Grafik für Joyce 64
Zum Schneider-Joyce wird die leistungsfähige Grafikschnittstelle GSX mitgeliefert. Mit einem Programm in Turbo-Pascal wird GSX einfach bedient

Erweiterter Bresenham-Algorithmus 66
Wir stellen zwei Prozeduren zum Zeichnen von Linien vor

MS-DOS-Interrupt 24H 72
Wer professionelle Programme in Turbo-Pascal schreibt, muß auch DOS-Fehlermeldungen abfangen. Wir beschreiben, wie das geht

Grafik mit EGA 78
Nach den ersten drei Teilen unserer EGA-Serie sollte die EGA-Karte für Sie keine Geheimnisse mehr bereithalten. Die mc bringt einen komfortablen EGA-Editor in Turbo-Pascal zum Abtippen

EGA-Editor

Software, die den Einsatz des EGA-Standards auf PCs erlaubt, ist sehr teuer. mc bringt einen komfortablen Grafik- und Texteditor für die EGA-Karte in Turbo-Pascal zum Nulltarif. Dabei wird genau beschrieben, wie das Programm funktioniert und wie man es an eigene Wünsche anpaßt.

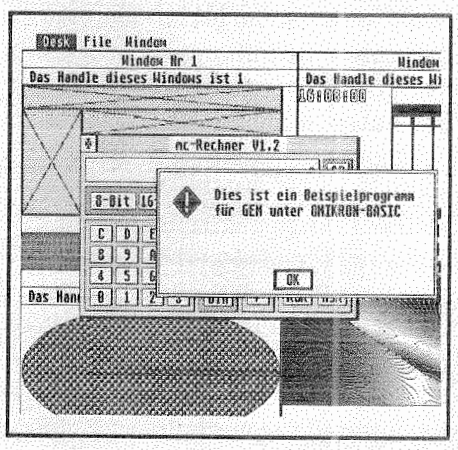
Titelfoto: Klaus Hager

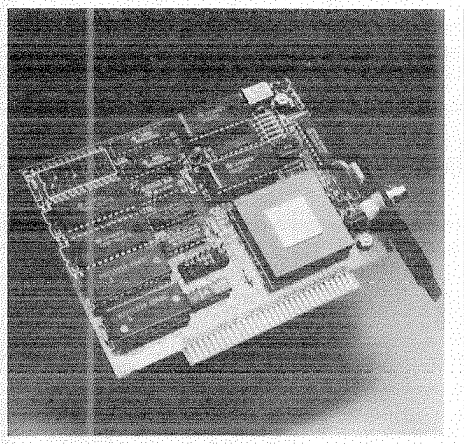
Seite 78

Omikron-Basic

Der halbcompilierende Basic-Interpreter für den Atari ST ist von Omikron in einem Steckmodul untergebracht worden. Techniker und alle, die genau und schnell rechnen müssen, werden sich über dieses Produkt freuen, das zu MBasic kompatibel ist.

Seite 104



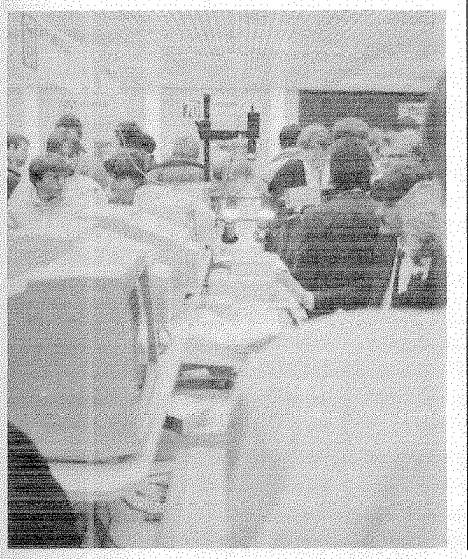


PC-Erweiterungskarten

Marktübersichten über PC-Erweiterungskarten hat es schon viele gegeben. Die mc zeigt als erste deutsche Zeitschrift 90 Karten mit ihren wichtigsten Daten und einem Foto. Dadurch wird das riesige Angebot im wahrsten Sinne des Wortes „übersichtlich". **Seite 49**

CeBIT 1987

Wird die CeBIT-Messe etwas oder bleiben Firmen und Besucher im Schmollwinkel? mc hat festgestellt, daß diese Messe das Eintrittsgeld wert ist, wenn man seinen Rundgang gut vorbereitet. Lesen Sie, was unser Marktbeobachter Reiner Schönrock bei den Großen der PC-Szene an Neuheiten gefunden hat. **Seite 12**



Command unter Turbo-Pascal 92
Registerübergabe- und Aufrufmechanismus als Inline-Statement verhindert Fehlfunktionen

Graphix-Toolbox und Fehlerbehandlung 94
Bei der Fehlersuche in Programmen hilft eine kleine Routine weiter

MS-DOS merkt sich 128 Befehle 96
Ein C-Programm zur Emulation der Unix-C-Shell

Apple nutzt neue Grafik-Betriebsarten 97
Einige Tips für Apple-Anwender

Symbolischer Disassembler für den Apple-II 99
Ein nützliches Programm, das auf den Standard-Disassembler aufsetzt

Siemens PC-D mit hoher Auflösung 101
Ein Programm, das mc für den Olivetti M-24 vorstellte, nun an den Siemens-PC-D angepaßt

mc-hard

Neue 32-Bit-Prozessoren von AT&T 67
Die neueste Prozessor-Familie von AT&T wartet mit interessanten Leistungsdaten auf

68020 im PC 91
Schnelle Arithmetik für Spezialisten

Automatischer Drucker-Umschalter 98
Eine kleine Schaltung übernimmt das Umstöpseln von Druckerkabeln

mc-test

Software für Grafiktablett 100
Für ein preiswertes Grafiktablett, über das wir bereits berichteten, gibt es jetzt ein gutes Treiberprogramm

Boeing-Calc 102
Dieses Programm ist für alle interessant, die bei Lotus 1-2-3 oder ähnlichen Programmen an Grenzen stoßen

Datensicherung bei Festplattenlaufwerken 103
Ein Streamer-System von Teac sichert 60 MByte in 10 Minuten

Omikron-Basic 104
Ein Basic-Interpreter mit überzeugenden Leistungsdaten hat im mc-test bewiesen, was er kann

mc-wettbewerb

Die fleißigen Biber 70
Gleichgültig, auf welchem Prozessor er in welcher Sprache sein Wesen treibt: der fleißigste Biber gewinnt einen Amiga 2000

mc-markt 110

mc-vorschau 150

Impressum 148

Zu wenig Information

Sie haben in Heft 2 /1987 den Apple-IIGS getestet. Können Sie mir sagen, wo in diesem Test – mit Ausnahme des kleinen Kastens auf Seite 96 – die Information steht? Sie schreiben, der Apple-IIGS wäre wahrscheinlich in C programmiert, weil er so langsam startet wie der Macintosh.
Mein Mac startet in 10 Sekunden. Das Betriebssystem des Macintosh ist in Pascal (und natürlich 68000-Assembler) geschrieben. Ich nehme dasselbe vom GS an.
Der GS hat im übrigen die gleichen Betriebssystemaufrufe wie der Mac. Jedes Hochsprachenprogramm läßt sich also mit minimalem Aufwand auf den GS übertragen. Dies läßt darauf schließen, daß Apple die Absicht hat, beide Rechnerfamilien näher zusammenzubringen. (Auch die 3½-Zoll-Diskettenlaufwerke lassen sich ohne Änderung an beiden Maschinen anschließen.)
Beim „GS" (Graphik und Sound) ist Ihnen ebenfalls ein Fehler unterlaufen: Die vier bzw. 16 Farben beziehen sich nämlich auf eine Rasterzeile. Insgesamt kann man mit Hilfe der Hardware 4096 Farben darstellen.
Der Rechner hat ein 15stimmiges Synthesizer-IC der US-Firma Ensoniq, das Signale aus dem Audio-Eingang des GS digitalisieren und verfremden kann.
Der Rechner ist im übrigen in der Lage, im Grafikmodus proportionale Zeichensätze auf dem Bildschirm darzustellen.
Der RGB-Ausgang ist analog. Die Schnittstellenstecker sind Apple-Standard und inzwischen fast überall zu bekommen (andere Firmen benutzen sie als Joystick-Stecker). Außerdem gibt es Adapterkabel.
Die Apple-IIe-Software läßt sich weiterverwenden, ebenso die Hardware. Eine Aufrüstung IIe nach GS sollte auch bald verfügbar sein. Diese Informationen stammen aus der Januarausgabe der US-Zeitschrift „MacUser".
Ich würde Ihnen folgendes vorschlagen: Überlassen Sie Artikel wie den angesprochenen der Bildzeitung (will sagen CHIP). Bringen Sie mehr Informationen, die jedem etwas bringen, insbesondere mir als Macintosh-Besitzer. Sie hatten früher sehr viel mehr Artikel von allgemeinem Interesse im Heft.
An der Zeitschrift mc stört mich die Verbindung von Information mit Meinung – etwa im Inhaltsverzeichnis die Überschrift „Grundlagen" für den IBM-PC-Bus-Beitrag (damit überhaupt was in jeder Rubrik erscheint?).

Matthias Urlichs
8501 Schwaig 2

Betrachtungen zum Thema Btx

Als privater Btx-Teilnehmer nutze ich das System hauptsächlich zur Kontoführung und -verwaltung und zur Geldüberweisung. (Erste Einschränkung: Dies kann auch durch direktes Anwählen des Bankcomputers erfolgen.)
Da ich als Terminal weder ein spezielles (teueres und für nichts anderes zu verwendendes) Btx-Gerät, noch ein modernes (und teueres) Btx-taugliches Fernsehgerät verwende, sondern einen C64/128, mit dem man eben auch „normal computern" kann, bin ich an Benutzergruppen angeschlossen, die Tricks, Tips und sogenannte Telesoftware anbieten. Damit eröffnet sich die Möglichkeit der Benutzung der „electronic mail" im Btx, das heißt die Verbindung mit Gleichgesinnten (C64/128), auch im erweiterten Gedankenaustausch des nicht-technischen Alltags, ebenso der Zugang zu allgemeinen Nachrichten, sofern sie kostenlos abrufbar sind. Das sind aktuelle Kurzmeldungen von Zeitungen und Nachrichtenagenturen (Reuters), Wetterbericht, Fernsehprogramme usw. (Die zweite Einschränkung: Diese könnten ohne Telefongebühr und Belegen der Telefonleitung auch über Videotext – hier Teledata – von einem entsprechend ausgerüsteten Fernsehgerät empfangen werden.) Schließlich können Tausch-, An- und Verkaufsangebote (Anzeigen) persönlicher Natur für neue und gebrauchte Artikel aller Art via Btx wahrgenommen werden.
Resümee: Für den privaten Nutzer ist Btx momentan nicht interessant, da die systemspezifische Dialog-Kommunikation von den Anbietern nicht ausgenutzt wird, bzw. durch zu geringe Teilnehmerzahl als nicht wirtschaftlich genug zur Einführung angesehen wird. Es fehlen echte Direktbestellungen von Waren des täglichen Bedarfs (Kaufhaus, Supermarkt, Großhandel).
Um z. B. einmal im Jahr ein Flugticket oder einen Mietwagen zu buchen, brauche ich kein Btx.
Da Btx eine relativ langsames Medium ist, wäre die Abschaffung (oder Änderung) des 8-Minuten-Taktes für Ortsgespräche durch Identifizierung der angewählten Btx-Nummer angebracht, d. h. die Gebühreneinnahmen werden durch Teilnehmeranreiz, nicht durch hohe Gebühren bestritten. Warum soll man nicht vom französischen Modell lernen?
Letzter Verbesserungsvorschlag: Die Ausweitung der „electronic mail"-Möglichkeiten innerhalb von Btx. Noch vor 20 Jahren wurde ein persönlicher Brief, der nicht handgeschrieben war, als unhöflich angesehen, heute akzeptiert man ohne weiteres Maschinengeschriebenes und morgen wird es „electronic mail" sein; auch an Empfänger, die nicht an Btx angeschlossen sind (Brieftelegramm von der Post). Außerdem ist eine Querverbindung ins internationale Telex (Teletex)-Netz anzustreben.

Zum Schluß: Die Aussage, daß die Pilotteilnehmer vorwiegend Leute mit technischem Interesse waren und demzufolge die Hochrechnung der Akzeptanz von Btx in der breiten Masse falsch ist, kann nicht unwidersprochen bleiben. Dies wird auch im Artikel „Btx ein Überlebenskampf" (mc 12/86) am Beispiel Frankreichs widerlegt. Oder sind Franzosen „technikfreundlicher"?

F.G. März
Rivonia 2128
Rep. Südafrika

MIDI-Schnittstelle

Die Applikation „MIDI-Interface für den Apple-II" von Joachim Lange hat mich sehr interessiert, nicht nur weil ich fast das gleiche für meinen CP/M-Computer, mit einem Dart als MIDI-SIO, aufgebaut und programmiert habe. (Dazu noch einen Compiler, der die MIDI-Daten in eine Notdatei, etwa so wie ich sie in mc-Heft 6/86 beschrieben habe, wandelt.)
Die verwendete Hardware (DX 27) entbindet Herrn Lange von dem Problem, im Receive-Programm zu entscheiden, welche Zeitspanne man als gleichzeitig im musikalischen Sinne betrachten kann. So wird bei gleichzeitigem Tastenanschlag nur das erste Statussignal „Keyon" übertragen und von den weiteren Tasten nur die Ordnungsnummer und der Velocity-Wert. Daten ohne vorangestelltes Statusbyte sind somit vom DX 27 für quasi-gleichzeitig erklärt. Mein Masterkeyboard von Boehm kennt diesen „Runningmode" nicht und gibt immer das Statusbyte mit aus. Bei manch anderen Fabrikaten mag das ebenso sein.
Im Receive-Programm behandle ich nun Events, die innerhalb 7 ms eintreffen, als gleichzeitig, d. h. der Zähler läuft weiter und es wird kein Zeiteintrag vorgenommen. Diesen Wert von 7 Millisekunden habe ich willkürlich gewählt und möchte ihn gern zur Diskussion stellen. Er soll auch für klassische Musik gelten. Vielleicht hat ein Leser einen plausiblen Vorschlag für einen bestimmten anderen Wert? *Horst Mohr*
5067 Kürten-Dörnchen

Betrug

Mich erschreckt die Resignation, die dem Leserbrief „Betrug" in mc 3/87 zu entnehmen ist! Damit sich auch „kleine" Betrügereien nicht lohnen, müssen sich die Betrogenen auch in solchen Bagatellfällen entschiedener zur Wehr setzen. Natürlich kann der Postbote nichts ausplaudern, aber die Frage nach der Gültigkeit des Nachsendungsauftrags gehört sicherlich schon zu seinen Aufgaben, auch wenn er nicht persönlich bekannt ist. Schließlich lag der Verdacht nahe, daß hier mindestens das Briefgeheimnis verletzt wurde. Ich halte sogar das Zurückhalten der Nachsendungsadresse für unberechtigt, da sie nicht von einem rechtmäßigen Auftraggeber angegeben wurde und damit auch nicht unter dem Schutz des Postgeheimnisses stehen kann. Oder stellt sich die Post hier schadenfroh stur, weil sie die Versendung von Bargeld in Briefen unterbinden will?

Bei geringer oder ungewisser Schadenshöhe fragt man sich natürlich, ob sich der Aufwand einer Strafanzeige bei der bekanntlich niedrigen Aufklärungsquote lohnt. Ich bin aber der Meinung, daß auch diese Unentschlossenheit zu der erschreckenden Anzahl von „Klein-Gaunern" beiträgt. Daraus folgt das heutige Mißtrauen jedem Mitmenschen gegenüber, mit dem ich mich auch nicht einfach abfinden mag. Außerdem könnte in diesem konkreten Fall sogar jemand den Anbieter verdächtigen, selbst den Nachsendungsauftrag gegeben zu haben, um für das Verschwinden von Geldsendungen ein perfektes Alibi zu haben.

Das einzige Mittel, sich zu wehren, ist zunächst eine Anzeige wegen Urkundenfälschung (Unterschrift auf dem Nachsendungsauftrag), wegen Diebstahls von brieflichen Nachrichten und wegen Verletzung des Briefgeheimnisses, da die Briefe sicher auch geöffnet wurden. Eine Anzeige wegen Diebstahls von Bargeld kann in dem Augenblick folgen, wenn sich der erste Besteller wegen der ausbleibenden Lieferung meldet. Damit hat solch ein Gauner schon einiges auf dem Kerbholz. Selbst wenn der Schwindel schon im Anfangsstadium aufgedeckt wird, bleibt für einen entsprechend höheren Geldbetrag der Vorwurf des versuchten Diebstahls.

Da nun mal vorbeugen besser ist als heilen, begrüße ich auch die Hinweise auf allgemein übliche Vorsichtsmaßnahmen. Schließlich dienen auch sie dem Ziel, den Betrügern das Handwerk zu legen. Aber Angstmacherei ist menschenunwürdig, und wenn jemand schon hereingelegt wurde, sollte er sich (auch gegenüber der Post) zu wehren wissen, statt sich den mitmenschlichen Umgang von schwarzen Schafen diktieren zu lassen.

R. Helmkamp,
6126 Brombachtal

Keine Telex-Zugangsmöglichkeit

Zu „Telex mit PCs", in mc 2/1987, Seite 34

Der Titel dieses Artikels hat mich „elektrisiert", der Inhalt jedoch ernüchtert, glaubte ich doch zunächst, daß Sie eine preisgünstige Telex-Zugangsmöglichkeit für alle PCs vorstellen. Dem Umfang des „Durchbruchs" entnehme ich aber, daß meine langwierige Suche nach einem entsprechenden Service noch lange nicht zu Ende ist. Zwar bieten viele Datex-P-Mailbox-Betreiber auch Telexzugang an, der aber ist mit NUI und Mailbox-grund-, -nutzungs- und den eigentlichen Telex-Gebühren für die private Anwendung recht teuer und geht über das Ausland, weil hierzulande eben nicht sein kann, was nicht sein darf.

Welchen (technischen und geographischen) Weg meine Texte zurücklegen, wäre mir ja egal, wenn ich sie nur direkt per Telefonleitung bei akzeptablem Preis via Gateway-Server senden und empfangen könnte.

Vielleicht animiert mein Brief jemanden, diese Marktlücke zu schließen (eventuell mittels Com-M-Tex und MS-DOS-Rechner?) oder auch einen vielleicht vorhandenen Anbieter, die Werbetrommel etwas hörbarer zu rühren!

Anselm Rapp
8000 München 71

Zeile fehlt

Ich möchte darauf hinweisen, daß in dem Programm des Beitrags „Turbo-Pascal bedient Hercules-Karte" (mc 2/1987, Seite 47) die Initialisierung der „6845" nicht durchgeführt wird. Es fehlt hier wohl die Zeile „PORT[Data Reg]:=Grp Val[Reg Sel]"., die nach der Zeile „PORT[Index Reg]:=Reg Sel" eingefügt werden muß. *Günter Villnow*
4170 Gelders 1

Anmerkung der Redaktion: Wir haben das schon in mc 3/1987 berichtigt. Ohne diese Änderung kann der Monitor zerstört werden.

Drucker-Versatz

Zur Frage von Herrn Lindecke (mc 12/86, Seite 12) habe ich folgende Antwort: Beim Einschalten von hoch- oder tiefgestellter Schrift beim SG-10 (und vielen anderen Druckern, z.B. bei meinem Gemini 10X) schaltet der Drucker automatisch auf Doppelanschlag um, daher der Versatz. Es gibt leider nur eine Abhilfe (zumindest ist mir bisher nichts besseres eingefallen): Zuerst eine Zeile mit den hochgestellten Zeichen drucken (alles andere durch Leerzeichen ersetzen), einen Wagenrücklauf (CR) ohne Zeilenvorschub (LF) senden, dann genauso die normalen und tiefgestellten Zeichen abschicken und zum Schluß auch noch den Zeilenvorschub hinterher.

Jan Hinnerk Haul
2000 Wedel

Korrektur zum „Pascal-Programm"

Das nützliche Programm im Artikel „Pascal-Programm leichter zu durchschauen" (mc 11/86) weist Fehler auf, die in einigen Fällen zu Komplikationen führen können: Nachdem das Programm eine Datei bis zum Ende bearbeitet hat, wird versucht, das Element „kommentare [0]" an das Dateiende anzuhängen. Da das nullte Element im Variablenteil aber nicht definiert wurde, werden zufällig im Speicher stehende Zeichen angehängt. Zur Abhilfe ist Zeile 14 im Programm wie folgt zu ändern:

```
kommentare: ARRAY
[0..schachtel-tiefe] OF
string40;
```

Außerdem muß zwischen Zeile 26 und 27 eine weitere Zeile eingefügt werden:

```
kommentare [0]:='';
```

Ein weiterer Punkt: find-first-keyword findet nur Schlüsselwörter, die in Kleinbuchstaben geschrieben sind. Da aber viele PASCAL-Programmierer Schlüsselwörter groß schreiben, sollte die Prozedur wie folgt geändert werden: Zwischen Zeile 69 und 70 folgende Zeilen einfügen:

```
n: integer;
zeile2 : string [120];
```

Zwischen Zeile 73 und 74 folgende Zeilen einfügen:

```
zeile2:=zeile;
FOR n:=1 TO length(zeile) DO
zeile2 [n]:=upcase (zeile[n]),
```

Zeile 74 und 75 wie folgt ändern:

```
ende:=(pos('PROZEDURE',
zeile2)+pos('FUNCTION',
zeile2)+pos('BEGIN',zeile2))
> 0;
```

Schließlich kann es in der Prozedur 'search' vorkommen, daß das Dateiende nicht richtig erkannt wird und dadurch laufend Leerzeilen in die Datei geschrieben werden. Das läßt sich vermeiden, indem man Zeile 100 wie folgt ändert:

```
IF (position>length (zeile))
AND NOT (eof (source)) THEN
```

Mit diesen Änderungen arbeitet das Programm einwandfrei. *Mario Zimmermann*
7370 Aichwald 4

CeBIT – die Messe der Messen

In Hannover fand vom 4. bis zum 11. März die weltgrößte Messe für Büro-, Informations- und Telekommunikation statt. Über 2400 Aussteller nutzten die sich jährlich bietende Chance, dem interessierten Fachpublikum ihre Neuheiten vorzustellen. Der Trend, der sich schon im letzten Jahr abzeichnete, wurde auch in diesem Jahr wieder deutlich unterstrichen: Neue Computer und Peripheriegeräte leisten mehr, sind deutlich kleiner geworden und kosten erheblich weniger. Die meisten Computerfirmen von A wie Apple bis Z wie Zenith stellten neue Computer mit erstaunlich niedrigen Preisen vor.

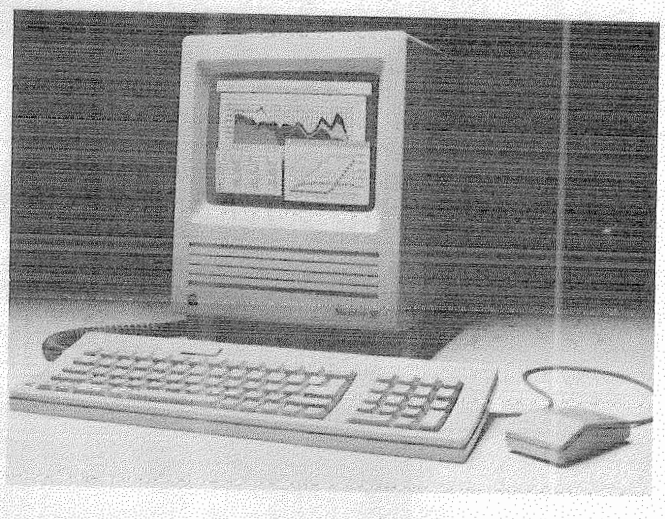


Bild 2. Der Macintosh SE hat eine Systemschnittstelle und kann MS-DOS-kompatibel gemacht werden

Apple: Zwei neue Modelle

Gleich zwei neue Modelle vom Macintosh konnte man bei Apple bestaunen, den Macintosh II (*Bild 1*) und den Macintosh SE (*Bild 2*). Der letztgenannte wird in zwei Versionen angeboten und ist im wesentlichen eine Erweiterung des Macintosh Plus mit der CPU 68000 (8 MHz), 1 MByte RAM (auf 4 MByte erweiterbar) und einem Systembus (SE-Bus), der für Erweiterungskarten gedacht ist. Für den 96poligen SE-Bus soll es in absehbarer Zeit eine MS-DOS-Steckkarte geben. Die Karte Mac86 von AST Research emuliert einen MS-DOS-Computer in einem Window auf dem Macintosh. Auf Mac86 laufen MS-DOS-Programme gleichzeitig mit Macintosh-Programmen. Beide Systeme können ein und dieselbe Festplatte benutzen. Für die Karte Mac86 ist auch ein 5¼-Zoll-Laufwerk erhältlich. An weiteren Steckkarten für Coprozessoren (68020, 68881), IEEE-488, größere Bildschirme oder diverse Kommunikationsnetze soll auch schon gearbeitet werden. Gegenüber dem Macintosh Plus sind die Macintosh SE-Modelle etwa 20 Prozent schneller; der ROM-Bereich wurde auf 256 KByte verdoppelt. Über den Apple-Desktop-Bus (ADB) können neben der Tastatur sieben weitere serielle Eingabegeräte angeschlossen werden. Für den Macintosh SE werden eine Standard- und eine MS-DOS-Tastatur angeboten. Die Version mit zwei Diskettenlaufwerken kostet 8900 DM, die mit einem Floppy-Laufwerk und einer 20-MByte-Festplatte 10 900 DM. Der Preis für den Macintosh Plus wurde nochmals um über 2000 DM gesenkt und liegt jetzt bei 6400 DM. Das Spitzenmodell von Apple ist der Macintosh II, der ab August für unter 19 000 DM zu haben sein soll. Dieses Modell arbeitet mit der CPU 68020 (16 MHz) und ist mit dem Coprozessor 68881 ausgestattet. Auf der Hauptplatine kann der Speicher des Computers, der mit 2 MByte RAM ausgeliefert wird, auf 8 MByte erweitert werden. Mit der optionalen PMMU 68851 können maximal 1,5 GByte RAM adressiert werden. Der Macintosh II hat sechs Erweiterungsplätze und eine 40-MByte-Festplatte integriert. Neben einer MS-DOS-Steckkarte soll es für diesen Computer auch eine Unix-Lösung geben. Ebenfalls von AST Research kommt die Karte Mac286, die auf dem Macintosh II den Einsatz von MS-DOS-Programmen erlaubt. Im MS-DOS-Window auf dem Bildschirm des Macintosh können auch CGA- oder Herkules-Grafiken angezeigt werden. Für den Macintosh II werden zwei Monitore angeboten. Auf dem monochromen 12-Zoll-Monitor sind durch einen speziellen Video-Chip 16 Graustufen darstellbar; mit einer Erweiterung des Video-Speichers sind maximal 256 Graustufen erreichbar. In der Farbversion wartet Apple mit einem 13-Zoll-Monitor auf, der 16 bzw. 256 Farben darstellen kann. Die Auflösung beider Monitore liegt bei 640 × 480 Pixel.

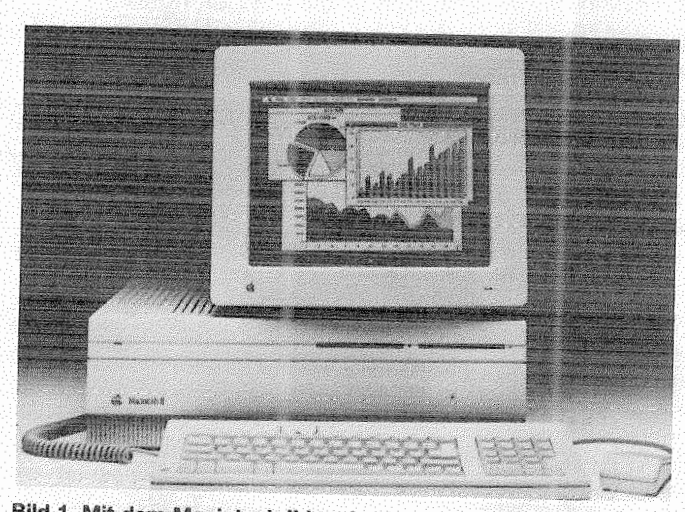


Bild 1. Mit dem Macintosh II hat Apple viele Wünsche der am „Mac" Interessierten erfüllt

Zur Unterstützung von Software-Entwicklern und Informatik-Lehrstühlen an Hochschulen wurde die Apple Programmers und Developers Deutschland (APD) gegründet.

Durch die Öffnung des Macintosh-Systems, gezielte Werbekampagnen und die ernormen Preissenkungen hofft Apple auf ein erfolgreiches Jahr 1987. Vom Macintosh SE möchte Apple in diesem Jahr etwa 100 000 und vom Macintosh II etwa 40 000 Einheiten verkaufen.

Atari: PC und Mega ST

Neben den neuen Computern der Mega ST-Serie, dem Laserdrucker (*Bild 3*) und dem Atari PC (*Bild 4*) konnte man bei Atari einen Vertreter der Heimcomputer der 800 XL-Klasse sehen. Da Atari mit diesem Modell recht guten Umsatz machte, soll das neue Modell 65 XE jetzt forciert werden. Das Hauptinteresse der Besucher galt allerdings den ST-Modellen und den zahlreichen Hard- und Software-Exponaten. Gleich mit drei Modellen erweiterte Atari die ST-Serie. Die Mega ST1 (1 MByte RAM), ST2 (2 MByte RAM) und ST4 (4 MByte RAM) sollen im Frühsommer 1987 eingeführt werden. Als Preise konnten bislang nur Größenordnungen angegeben werden. Der Mega ST1 soll etwa 2500 DM kosten, der Mega ST2 etwa 3000 DM und der Mega ST4 etwa 4000 DM. Äußerlich unterscheiden sich die neuen Modelle von den 520 ST- und 1040 ST-Modellen durch eine abgesetzte Tastatur und ein kompaktes Gehäuse mit einem eingebauten Floppylaufwerk. In den Mega ST-Modellen ist der Systembus über eine Steckleiste für Erweiterungen zugänglich. Auf der Hauptplatine der neuen Modelle ist der Blitter-Coprozessor, der für schnellere Grafikverarbeitung sorgt, und eine batteriegepufferte Echtzeituhr integriert. Am Betriebssystem wurde laut Atari außer

den Blitter-Routinen nichts geändert, die neuen Modelle sollen voll kompatibel zu den bewährten ST-Modellen sein. Für diese soll der Blitter-Chip mit sechs neuen ROM-Modulen ab August als Upgrade verfügbar sein.
Im gleichen kompakten Design der Mega ST-Modelle wird eine 20 MByte-Festplatte (SH 205) angeboten. Der Preis für die recht lautstarke Festplatte SH 204 wurde um 700 DM auf 1298 DM gesenkt. Noch in diesem Jahr soll es ST-Modelle mit mehr Farben und einer Auflösung von 1280 x 960 Punkten geben. Auch an 68020-Computern wird bei Atari gearbeitet.
Eine weitere Neuheit auf dem Atari-Stand auf der CeBIT war der Laserdrucker SLM, der bis zu acht Seiten in der Minute druckt und unter 3000 DM kosten soll. Im Paket mit dem Mega ST2 soll der Preis inklusive unterstützender Software unter 6000 DM liegen. Auch an die älteren ST-Modelle mit 1 MByte RAM soll der Laserdrucker angeschlossen werden können. Der Drucker erreicht eine Auflösung von 300 Punkten pro Zoll und kann alle Zeichen und Grafiken, die auf dem Bildschirm dargestellt werden können, ausdrucken. Wann der Laserdrucker verfügbar ist, steht noch nicht fest. Während des Ausdruckes kann mit dem Computer weitergearbeitet werden, da der Betrieb des Laserdruckers nur etwa 10 Prozent der Rechnerleistung benötigt. Je nach Zeichensatz belegt der Laserdrucker den Speicherplatz des Computers mit, da in dem Laserdrucker selbst nur die Mechanik untergebracht wurde.
Mit dem Atari PC erweitert Atari sein Angebotsspektrum an Computern in ein Marktsegment, das schon von vielen anderen Herstellern abgedeckt wird. Der Atari PC wird im Sommer 1987 zu einem voraussichtlichen Preis von 1798 DM eingeführt werden. Im Design ähnelt er den Modellen der Mega ST-Serie. Einziger Unterschied zu diesen Computern ist, daß das 5¼-Zoll-Floppylaufwerk auf der linken Seite eingebaut ist. Neben einem Hauptspeicher von 512 KByte enthält der Atari PC einen Bildschirmspeicher von 256 KByte. An Grafik-Karten emuliert der Atari PC neben dem Hercules-, dem MGA- und dem CGA- auch den EGA-Standard. Die 8088-CPU kann wahlweise mit 4,77 MHz oder 8 MHz betrieben werden. An Peripheriegeräten sind die gleichen Systeme anschließbar, die auch von den ST-Modellen her bekannt sind. Besonders interessant ist der Einsatz des 3½-Zoll-Laufwerkes, das sich als sehr zuverlässig erwiesen hat. Einige auf der CeBIT durchgeführten Tests zeigten, daß der Atari PC voll kompatibel ist und auch im 8-MHz-Modus ohne Probleme arbeitete. Wo die meisten Anbieter von Low-Cost-PCs sparen, hat Atari den vom ST her gebotenen Qualitätsstand gehalten – beim Monitor. Auch Grafiken in der höchsten Auflösung (720 × 348 Punkte) sind klar und deutlich zu erkennen und stellen die Produkte vieler Mitbewerber in den Schatten.

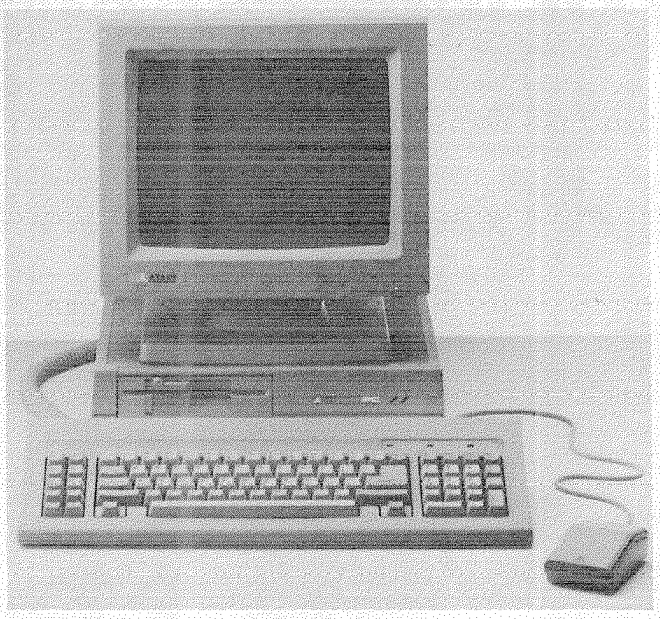


Bild 4. Mit dem Atari PC kommt ein preiswertes, IBM-PC-kompatibles System auf den Markt

Die ersten Programme für den Atari PC (und für alle anderen kompatiblen) wurden von GFA angekündigt, deren GFA-Basic ein Standard bei den ST-Anwendern geworden ist. Mit GFA-Fakt (149 DM) zur Fakturierung und Lagerverwaltung und GFA-Desk (99 DM) zur kombinierten Textverarbeitung mit Adreßverwaltung sind zwei Niedrigpreis-Programme verfügbar. GFA kündigte die Version 3.0 des GFA-Basic für den Atari ST an, die 100 neue Befehle für GEM- und Mathematik-Programme enthalten soll. Außer über eine Mailbox (0211/588-013/-014) kann man bei GFA jetzt auch über einen Club (24 DM Jahresbeitrag) an alle Neuheiten herankommen.

Bild 3. Links der Atari Mega ST2 mit einer 20-MByte-Festplatte SH 205, rechts der Atari-Laserdrucker

Die Version 2.0 von GFA-Basic ist bei Weide-Elektronik als Modul (198 DM) zu beziehen.
Für den professionellen Anwender bietet Rhothron mit dem Rho-Bus-System (dem Bus des mc-68000-Computers) eine günstige Lösung zur nicht vorhandenen Systemschnittstelle in den ST-Modellen an. Über einen Adapter, der auf die CPU gesteckt wird, wird eine VG-Steckerleiste mit acht Steckplätzen verfügbar, über die zahlreiche Steckkarten mit dem Atari ST betrieben werden können. Von der 2-MByte-RAM-Erweiterung bis zum VME-Bus-Interface bietet sich alles, was der industrielle Anwender benötigt. Neben dem VME-Bus bietet GTI aus Berlin jetzt einen ECB-Bus-Controller mit entsprechenden Erweiterungskarten an.

Commodore: Amiga 500, 2000, PC-I

Erwartungsgemäß standen die neuen Modelle des Amiga, über die wir in der letzten Ausgabe schon ausführlich berichteten, im Mittelpunkt der Präsentation von Commodore auf der CeBIT. Mit der Abkürzung OSA, die für Open System Architecture steht, wirbt Commodore für den Amiga 2000, der in der Grundversion ohne Monitor 2995 DM kostet. Der Amiga 500 für 1298 DM wird von Commodore als der Heimcomputerstandard der 90er Jahre propagiert. Auch für den kleinen Amiga soll es den MS-DOS-Emulator Sidecar geben. Die bislang für den Amiga 1000 verfügbare Lösung (1995 DM) kann am Amiga 500 nicht betrieben werden, da der Systembus gegenüber dem Amiga 1000 um 180 Grad gedreht ist. Neu war in diesem Zusammenhang die Aussage von Winfried Hofmann, dem Geschäftsführer Commodore Deutschland, daß der Amiga 1000 als eigenständiges Produkt weiter unterstützt werden soll. Erweiterungen für

dieses Amiga-Modell wurden angekündigt. Erstmals konnte ein Katalog über 500 Hard- und Software-Produkte zum Amiga vorgelegt werden, der gegen einen Unkostenbeitrag von 10 DM abgegeben wurde. Neben der aufwendigen Präsentation des „Amiga-Kosmos" stellte Commodore den PC 40/40 vor. Dabei handelt es sich um einen AT mit umschaltbarer Taktfrequenz von 10 MHz auf 6 MHz und einer 40-MByte-Festplatte mit einer Zugriffszeit unter 30 Millisekunden. Die in diesem Computer enthaltene Grafikkarte kann sowohl EGA, als auch CGA, MGA und Herkules-Grafiken verarbeiten. Die Preise aller Commodore-PCs wurden kräftig gesenkt. So kostet der PC II mit 3995 DM jetzt 2000 DM weniger als vor der CeBIT. Neben dem PC 10 II, der jetzt für 2995 DM zu haben ist, stellte Commodore einen Low-Cost-PC, den PC I vor (*Bild 5*). Dieser PC soll bei seiner Markteinführung im Sommer 1987 exklusive Monitor 1295 DM kosten. In dem Gerät ist ein 5¼-Zoll-Floppylaufwerk integriert. Die 8088-CPU ist mit 4,77 MHz getaktet, der Hauptspeicher mit 512 KByte ausgelegt. Der PC I wurde im Braunschweiger Werk von Commodore entwickelt und wird dort auch zusammengebaut. Neben der parallelen und der seriellen Schnittstelle bietet der PC I einen RGB- und einen Composite-Video-Anschluß. Für den herausgeführten Systembus soll es in absehbarer Zeit eine Erweiterungsbox für den Ausbau des Systems geben. Alle PCs und ATs von Commodore werden ab sofort mit MS-DOS 3.2 ausgeliefert.

Für den erfolgreichen Heimcomputer C-64 wurde eine 20-MByte-Festplatte und eine Speichererweiterung von 265 KByte vorgestellt. Die Einführung der Festplatte wurde von der Nachfrage abhängig gemacht.

Weitere Neuheiten

Neben den für die deutsche Computerszene interessantesten Unternehmen Apple, Atari und Commodore hatten natürlich sehr viel mehr Aussteller auf der CeBIT neue Computer, Drucker und andere Produkte zu zeigen, die wir unseren Lesern nicht vorenthalten wollen. In dieser und der nächsten Ausgabe finden Sie in den Rubriken mc-info und mc-markt Berichte über die Neuheiten der diesjährigen CeBIT. Hier nur noch das wichtigste in Kürze:

Zenith zeigte den Z-386, der mit 16 MHz ohne Waitstates arbeitet. In einem speziellen Speicherbereich mit 35 Nanosekunden Zugriffszeit werden systemresidente Routinen verarbeitet. Unter MS-DOS können mit diesem Computer bis zu 8 MByte RAM verwaltet werden. Für 2995 DM bietet Zenith den 8088-Computer Z-159 an, der in der Grundversion mit 768 KByte RAM und EGA ausgestattet ist.

Von Multitech kommt der mit 4,77/10 MHz getaktete 8088-Computer Multitech 710, der on board auf 768 KByte RAM erweitert werden kann und 1145 $ kostet. Neben einem 68020-Unix-System stellte Multitech einen 80386-Computer für 12 995 DM vor, der von 4,77 MHz bis 16 MHz stufenweise getaktet werden kann. In dem Preis ist eine 40-MByte-Festplatte mit 28 Millisekunden Zugriffszeit, 1 MByte RAM, ein 1,2 MByte-Floppylaufwerk, ein MGA-/CGA-Controller und ein 14-Zoll-Monitor enthalten. Ohne Festplatte wird dieses 80386-System für 10 995 DM angeboten. Mit einer Taktfrequenz von 20 MHz ist der 80386-PC von der Fast-GmbH aus München für etwa 12 540 DM noch schneller. Noch preiswerter bekommt man einen 80386-PC von Asscodat, dem Exklusivdistributor von Apricot-Computern. Der Apricot XEN-i 386/30 mit 1 MByte RAM und einer 30-MByte-Festplatte kostet 11 400 DM.

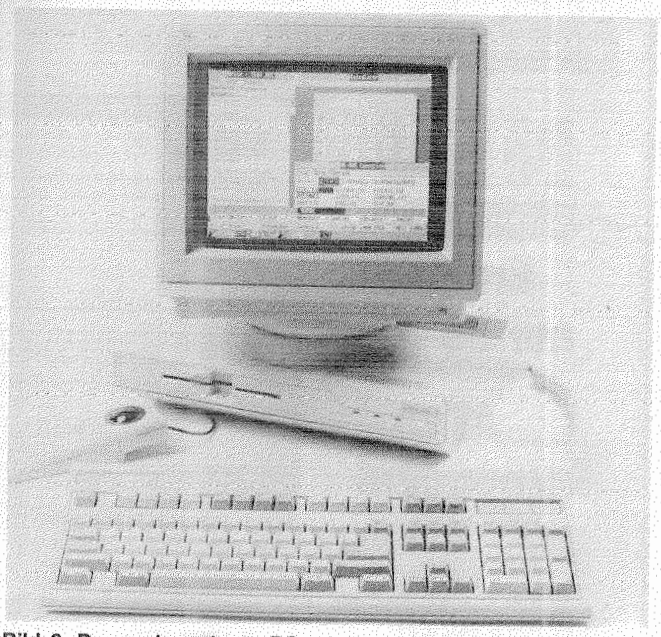


Bild 6. Der preiswerteste PC mit einer 80386-CPU kommt von Apricot und ist kompatibel zum IBM AT

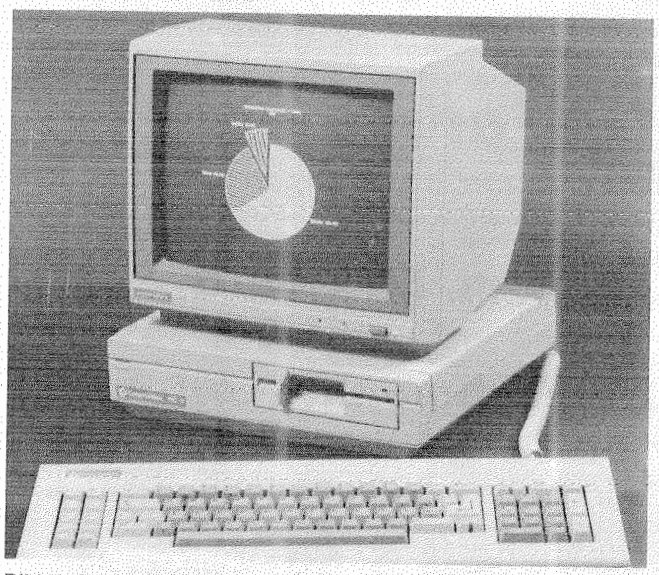
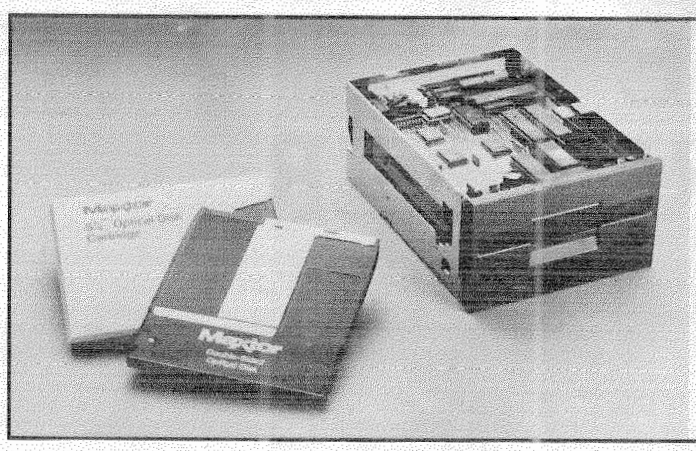


Bild 5. Commodore lehrt die Konkurrenz das Fürchten: Nur 1295 DM kostet der PC I

Für den Basic-Fan gab es bei Heimsoeth das Turbo-Basic zu sehen. Das zu Basica und GW-Basic kompatible Basic kompiliert etwa 12 000 Zeilen in der Minute und unterstützt den EGA-Standard voll. In diesem Basic dürfen Strings 32 KByte lang sein. Das angekündigte Turbo-C war noch nicht verfügbar, soll aber in absehbarer Zeit kommen.

Mehrere Firmen, darunter Olivetti, Sharp und NEC, stellten interessante Lap-Top-PCs vor. Von NEC kommt eine 2,4 GByte Festplatte, die in 9-Zoll-Technik ausgeführt ist und über die SCSI-Schnittstelle bedient wird. Die Formscan GmbH aus Königstein zeigte einen Seitenleser für 33 800 DM, der acht Schriftarten erkennen kann. Bei Citizen gab es neben 24-Nadel- und dem Typenraddrucker Premiere 35 den Laserdrucker Overture 110 für 6 500 DM zu sehen, der 10 Seiten in der Minute schafft und mit serieller und paralleler Schnittstelle ausgerüstet ist. Für 6270 DM gibt es den PagePrinter 8 von Centronics, der acht Seiten in der Minute schafft. Genauso schnell ist der Canon LBP-8 II. Das Spitzenmodell von Hewlett-Packard, der Laserjet 2000, schafft 20 Seiten in der Minute. Dieses Modell hat 34 Zeichensätze eingebaut und bietet 1,5 MByte RAM.

Bei allen Preisen, die wir in diesem Artikel genannt haben, handelt es sich um unverbindliche Preisempfehlungen der Hersteller inklusive Mehrwertsteuer. Da die meisten Produkte aber erst im Frühsommer 1987 zu haben sein werden, ist mit Preisänderungen zu rechnen. Alle Anbieter werden den Markt intensiv beobachten und die Preise ihrer Produkte entsprechend anpassen.



Laufwerk für optische Platten von der Firma Maxtor

Gemeinsames Aufzeichnungsformat für optische Speicher

Amerikanische und japanische Firmen bemühen sich derzeit, sich auf ein gemeinsames Aufzeichnungsverfahren für optische 5,25-Zoll-Disketten zu einigen. Die größten Chancen scheint das sogenannte Composite/Continuous-Format zu haben. Das Prinzip: Wie bei CD-Plattenspielern wird der Kopf ständig nachgeführt. Damit lassen sich auf jeder Seite der optischen Platte mindestens 300 MByte unterbringen. Das Verfahren ist selbsttaktend und basiert auf der RLL-Codierung, die auch bei magnetischen Festplatten verwendet wird. Zur Fehlerkorrektur wird man wahrscheinlich den Long-Distance-Reed-Solomon-Code benutzen. Ihn unterstützt auch eine Reihe von Bauelemente-Herstellern.

In den maßgeblichen Normungsgremien – X3B11 des American Accredited Standards Committee und SC-23 des Japanese National Committee – hat sich eine Mehrheit für das Composite/Continuous-Format herausgebildet. Firmen wie Maxtor, Hewlett-Packard, Philips, Western Digital, Hitachi, Fujitsu und NEC gehören zu seinen Befürwortern. Eine kleine Gruppe von Herstellern favorisiert allerdings das sogenannte Sampled-Servo-Format, bei dem auf der Platte Daten- und Servozonen getrennt sind.

Spezial-Netz für Schulen

Ein Spezial-Netz für Computer im Unterricht hat die Firma Triumph-Adler unter der Bezeichnung TA-PCNet vorgestellt. Es ermöglicht einen Dialog zwischen Lehrer und Schülern und berücksichtigt didaktische und pädagogische Anforderungen. Jeder Computer-Bildschirm wirkt dabei wie eine „elektronische Tafel", an der individuell verschiedene Aufgaben bearbeitet werden können. Die Bildschirminhalte lassen sich unter den einzelnen Stationen austauschen.

Bis zu 32 Personal Computer unterschiedlicher Leistungsklassen sind über das Netz miteinander verbunden – einer davon dient als Zentrale. Lehrer und Schüler haben uneingeschränkten Zugriff auf die zentral gespeicherten Programme und Daten. An jedem der angeschlossenen Arbeitsplätze können bis zu vier verschiedene Programme gleichzeitig ablaufen und in Fenstern auf dem Bildschirm beobachtet werden. Dem Lehrer steht ein besonderes Programm zur Verfügung, mit dem er die Verbindung zu den Schüler-Arbeitsplätzen herstellt. Somit kann er Kontakt mit jedem einzelnen Schüler aufnehmen, indem er dessen Bildschirminhalt auf seinen eigenen Monitor überträgt. Er kann sich aber auch in das laufende Schülerprogramm einblenden und beispielsweise seine Hilfe anbieten, falls er bemerkt, daß der Schüler einen falschen Lösungsweg eingeschlagen hat. Eine weitere Möglichkeit ist die Konferenzschaltung, bei der der Lehrer seinen Bildschirminhalt auf die Schüler-Bildschirme (oder eine ausgewählte Gruppe) überträgt. Während der Konferenzschaltung werden die laufenden Programme der Schüler angehalten; sie können danach dort fortgesetzt werden, wo sie unterbrochen wurden.

C-TOOLS

Die C-Tools sind nicht nur *direkt einsetzbar* (für **die meisten C-Compiler** z.B. Lattice-C und Microsoft-C), sondern verstehen sich auch als Know how-Tools:

Ausführliche *Begleitdokumentationen* liefern Ihnen detaillierte Informationen und erklären jedes Statement der C-TOOLS.

C-TOOLS Package # 1: Routinen für den Zugriff auf *sämtliche Systemeinheiten* von IBM-Personalcomputern und Kompatiblen, auf die Funktionen des *ROM-BIOS* und des Betriebssystems *DOS* für die Programmiersprache C im **deutsch kommentierten** *Source-Code* und im *Objekt-Code*.

Das 1. Package der C-TOOLS enthält über 100 Zugriffsroutinen auf Platte, Bildschirm, Tastatur, Drucker, Lautsprecher, den asynchronen Kommunikationsadaptor und weitere Tools. Soweit möglich, werden die Zugriffsroutinen jeweils auf *allen 3 Zugriffsebenen* zur Verfügung gestellt: auf Programmiersprachen-Ebene in C, auf der Betriebssystem-Ebene v. DOS u. auf BIOS-Ebene. Für alle BIOS-Zugriffe gibt es *assemblersprachliche Schnittstellen*, die auch mit anderen Programmiersprachen verwendet werden können.

Außerdem werden Ihnen unterschiedliche Verfahrenstechniken erklärt, z.B. für *schnelle, störungsfreie Bildschirmausgaben, Bildschirmfenster, Scrolling* etc.; Sie erhalten ein *Synthesizer-Programm*, mit dem Sie auf Ihrer Tastatur beliebige *Tonmuster* oder *Melodien* spielen und diese dann direkt in Ihr Programm einbauen können; Sie erhalten *Druckroutinen* für millimetergenaues Drucken in Vordrucke und für Graphik-Drucken u.v.m.

Preis: 632,70 DM

C-TOOLS Package # 2:

Datenorganisation und Speicherkonzepte, Sortierverfahren, Suchverfahren, Filter für die Programmiersprache C im deutsch kommentierten Source- und Objektcode.

Das Package # 2 enthält in der vorliegenden Version Routinen für:

- interne Datenorganisation/Speicherkonzepte: Listen, Stacks, Hashing inclusive aller Grundoperationen (z.B. Element einfügen, löschen, Position ermitteln etc.).
- Dateiorganisation und -zugriffe: sequentielle, "hashed" und indizierte Dateien
- Arbeitsspeicher- interne, externe und intern/extern-kombinierte Sortierverfahren
- Filter (z.B. variable lexikalische Sortierung, Dupletten-Filter, Dateiverschlüsselung, Spaltenanordnung etc.)

Preis: 855.– DM

C-TOOLS Package # 3: Ein Generator für dialogorientierte Programmsysteme incl. **Windowing**.

Die überwiegende Anzahl von Anwendersystemen ist heute dialogorientiert. Die Gestaltung der Benutzerschnittstelle ist in erster Linie verantwortlich für die sog. Benutzerfreundlichkeit eines Programmsystems und bestimmt damit maßgeblich dessen Marktchancen. Die Gestaltung d. Benutzerschnittstellen wird deshalb immer trickreicher u. komfortabler. Die Programmierung solcher dialogorientierter Programmsysteme verlangt jedoch dem Programmierer einen großen Aufwand an Zeit u. Arbeit ab. Diese Programmierarbeiten erheblich zu reduzieren, ist die Aufgabe eines Dialogsystem-Generators.

Der Dialogsystem-Generator kann: Graphik- u. Textmodus, Manipulation der Bildschirmattribute, Windows/Pull-Down-Menues, Ein-/Ausgabefelder, Cursormanagement, Dialog- u. Aktionssteuerung. **Preis: 855.– DM**

C-Tools Package # 4: Graphik

(enthält z.B. Line, Scale, Draw, Circle, Fill, Pattern, 2D-Transform, Window, Segment, Smooth, File, Calculation, Projection, Text, Hidden-Line)

Das Package enthält zu jedem Gliederungspunkt mehrere C-Funktionen im Quell- und Objektcode mit lauffähigen Demo-Programmen. Derzeit werden der Farbgraphik-Adaptor und die Hercules-Karte unterstützt.

In Kürze erfolgt ein Update für EGA, Olivetti u.a.

Preis: 855,– DM

C-Trainer

Lernen Sie C richtig von Anfang an!

Besonders geeignet für Schulungszwecke und C-Anfänger (C-Interpreter mit Tutorial-Programmierbuch – ein kompletter C-Lernkurs). Der C-Trainer ist eine neue, sehr effektive Methode, um C zu lernen oder sein Wissen zu erweitern. Der C-Trainer besteht aus drei Teilen: Turorial-Buch, C-Interpreter und eine C-Programmbibliothek.

Der C-Interpreter erlaubt eine hervorragende Kontrolle über die Ausführung eines C-Programmes und besitzt große Vorteile bei der Entwicklung von C-Programmen. Das Programm kann an einem beliebigen Punkt gestoppt werden, die Werte aktiver Variablen können eingesehen und geändert werden.

Programmänderungen sind sofort und ohne Compilieren und Linken möglich.

Separater oder gemeinsamer Trace für Funktionsaufrufe, Statements und Expressions ist möglich.

Verfügbar für: IBM/PC 285,– DM, Macintosh 285,– DM, Sun 513,– DM, MicroVAX (Unix o. VMS) 513,– DM, Pyramid 1425,– DM, VAX 11/700 (Unix o. VMS) 969,– DM, C-Tutorial-Buch 58,85 DM

(Alle Preise zzgl. Verpackung und Versand).

Der C-Trainer ist ein Produkt der Calalytix Corp.

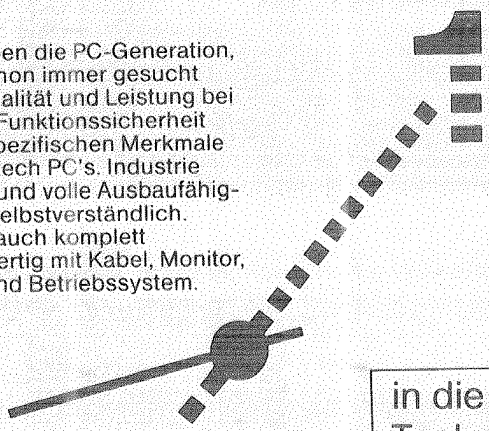
Die C-Tools sind Produkte des:

ECO Institut, Postfach 1158, D-8411 Lappersdorf, Telefon (0941) 82509

Multitech

...und weiter geht es

Wir haben die PC-Generation, die Sie schon immer gesucht haben: Qualität und Leistung bei optimaler Funktionssicherheit sind die spezifischen Merkmale aller Multitech PC's. Industrie Standard und volle Ausbaufähigkeit sind selbstverständlich. Natürlich auch komplett anschlußfertig mit Kabel, Monitor, Tastatur und Betriebssystem.

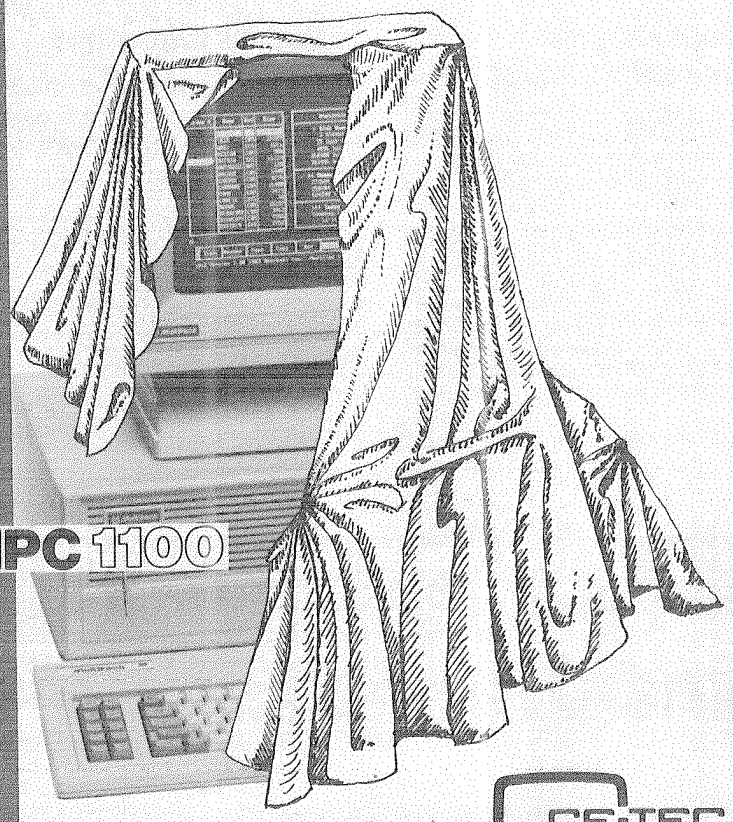


ACCEL 900 AT und die bewährten Produkte in ausgereifter Technik. Jetzt bei Multitech neu im Programm:
- Laser-Printer
- Comocolan Netzwerk

Die neue Technologie des MPC 1100 mit 32 BIT-Rechner zu erschwinglichen Preisen.

in die Technologie des 80 386

Wir lüften das Geheimnis!



MPC 1100

C46153

Generalimporteur: CE-TEC International

Bitte Händlernachweis anfordern.

CE-TEC Trading GmbH · Kornkamp 4 · D-2070 Ahrensburg
Telefon: 04102/49 01-0 · Telex: 2 189 875 · Telefax: 04102/49 01 38

mc-info

X/Open-Gruppe richtet Portabilitäts-Zentrum ein

Eines der dringendsten Probleme der Software-Industrie ist die mangelnde Übertragbarkeit (Portabilität) von Software. Anders ausgedrückt: Software, die auf einem Computer läuft, funktioniert auf einem anderen nicht. Eine Initiative, dieses Problem aus der Welt zu schaffen, ist die X/Open-Gruppe. Ihr Ziel ist es, auf der Basis von Unix eine gemeinsame Umgebung für Anwendungsprogramme zu definieren. Der Kern dieser Übereinkunft wurde inzwischen in Form des „X/Open Portability Guide" veröffentlicht. Folgende Firmen gehören dieser Initiative bisher an: Bull, DEC, Ericsson, Hewlett-Packard, ICL, Nixdorf, Olivetti, Philips, Siemens und Unisys.

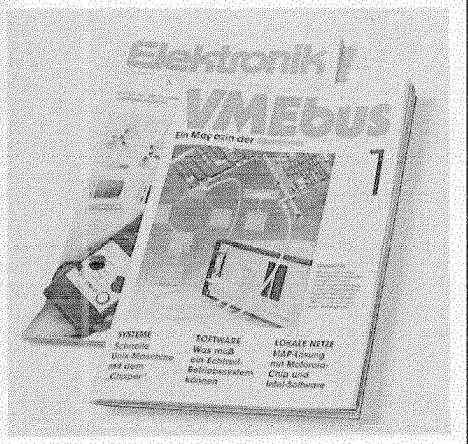
Diese Gruppe wird Software-Produzenten bald einen besonderen Service bieten: In der Nähe des Londoner Flughafens Heathrow entsteht derzeit ein Portabilitäts-Zentrum, in dem Hardware aller Mitgliedsfirmen zu finden ist. Wer diesen Service in Anspruch nimmt, kann an einem Ort seine Software auf all diesen Maschinen testen. Spezialisten sind dabei behilflich. Dies soll nur der erste Schritt sein, denn weitere solche Zentren werden folgen.

KI in der Fertigung erfolgreich

Eine Studie des Marktforschungsunternehmens Frost & Sullivan beschäftigt sich mit dem Einsatz der künstlichen Intelligenz (KI) in US-Fertigungsbetrieben. Sie geht davon aus, daß im Jahr 1986 in den USA KI-Produkte für rund 425 Mio. $ abgesetzt wurden. Bis 1990 soll sich das Volumen auf das Vierfache ausweiten. Dabei sind Roboter, bild- und sprachverarbeitende Systeme noch gar nicht berücksichtigt. Die Fertigung, so die Studie, biete das höchste Wachstumspotential für die künstliche Intelligenz, da diese Disziplin dort für sehr viele Aufgaben eingesetzt werden könne. 1990, so schätzt man, werden 40 % des amerikanischen KI-Absatzes auf die Fertigungsindustrie entfallen – im Jahr 1985 waren es nur etwa 25 %. In konkreten Zahlen ausgedrückt: Man geht davon aus, daß 1986 etwa 145 Mio. $ von Fertigungsbetrieben in die KI investiert wurden; bis 1988 soll sich der Betrag verdoppeln, um im Jahr 1990 die 700-Mio.-$-Grenze zu überschreiten.

Wissen aus dem Franzis-Verlag

Ob man Probleme mit einem Schneider-CPC, einem IBM-PC oder der Datenfernübertragung hat, sich in eine Programmiersprache oder in den Aufbau musikalischer Effekt-Geräte einarbeiten will: Kaum ein Bereich der Elektronik, in dem nicht Sonderhefte aus dem Franzis-Verlag genau die gesuchte Information bieten. Auch eine neue Zeitschrift bringt der Franzis-Verlag jetzt auf den Markt: das VMEbus-Magazin. Das erste Heft ist bereits im März erschienen. In diesem Jahr sollen drei weitere Ausgaben folgen. Ab 1988 ist eine zweimonatliche Erscheinungsweise geplant.



Der VMEbus, in Deutschland bei Motorola erfunden und inzwischen international genormt, ist ein 32-/16-Bit-Bussystem, für das über 200 Hersteller Produkte anbieten. Insbesondere in der Industrieautomatisierung ist diese Entwicklung sehr erfolgreich.
In diesem Jahr kostet das Einzelheft des neuen Spezial-Magazins 10 DM und das Jahresabonnement 30 DM. Bevor man also das Rad noch mal erfindet, sollte man einen Blick in das Angebot von Franzis werfen – es könnte eine Menge Arbeit sparen!

APL für 68000-Maschinen

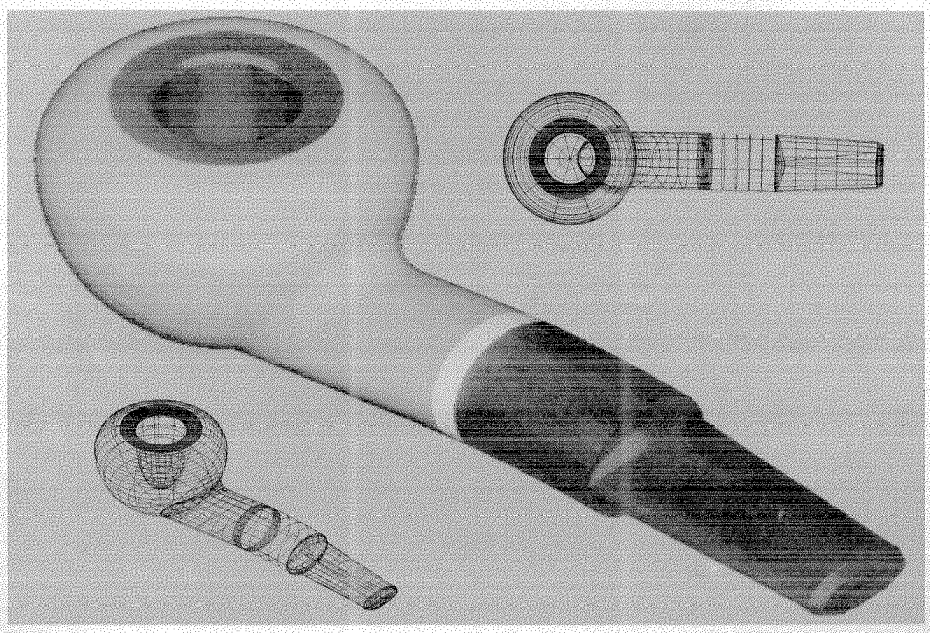
APL ist eine Programmiersprache, die schon in den sechziger Jahren bei IBM entwickelt wurde. Doch erst jetzt scheint es für sie die richtige Hardware zu geben. Sie verlangt nämlich eine sehr hohe Rechenleistung. Mit seinem großen linearen Adreßraum eignet sich der Prozessor 68000 ideal für diese arrayorientierte Sprache. In APL lassen sich beispielsweise Matrizen von der Komplexität eines Spreadsheets per Direkteingabe ganz oder in Ausschnitten manipulieren.
Die Bielefelder Firma GDAT bietet für den Macintosh von Apple sowie für die neueren Maschinen Atari-ST und Commodore-Amiga einen APL-Interpreter an, der vollständig in Maschinensprache geschrieben ist. Die derzeitige Version wurde seit 1981 ständig optimiert. Sie ist kompatibel zur IBM-Großrechnerversion. Routinen in anderen Sprachen lassen sich problemlos in Programme einbinden, die mit dieser APL-Version erstellt wurden. Index-sequentieller Zugriff (ISAM) ist im Sprachumfang bereits enthalten. Die Dateihandhabung ist ähnlich einfach wie der Umgang mit GEM-Dateien oder dem Finder des Macintosh.
Die ST-Version des Interpreters arbeitet unter GEM und bietet umfassenden Zugriff auf dessen Grafik, auf Dialogboxen, Menüs und Dateien. Auch die Maus-Funktionen sind programmierbar. Entsprechendes gilt für den Macintosh und den Amiga. In der ST-Ausführung kostet das Programmpaket 698 DM (inkl. MwSt.).

Xenix auf Vordermann

Eine Vereinbarung mit AT & T ermöglicht es jetzt der Firma Microsoft, ihr Betriebssystem Xenix auf den neuesten Stand des Unix-Systems-V (Release 3) zu bringen. Das Unternehmen will in Zusammenarbeit mit großen Kunden und Entwicklungspartnern so schnell wie möglich eine entsprechende Xenix-Version für die Prozessoren 80286 und 80386 herausbringen. Einige der renommierten Kunden sind IBM, Compaq, Altos und Tandy.
Xenix ist bisher über 175 000mal installiert worden, öfter als jeder andere Unix-Abkömmling. Die neueste Version soll abwärtskompatibel zu den Vorgängern sein, damit bisherige Programme auch auf dem neuen Betriebssystem laufen.
Microsoft wird außerdem ein Toolkit für den 80386 anbieten, das z. B. einen C-Compiler, einen Makroassembler und einen Debugger enthalten wird. Diese Werkzeuge sind insbesondere auch für Besitzer des neuen Deskpro-386 interessant.

Wunschpfeife per Computer

Mit Hilfe des Computer Aided Design will Deutschlands größte Pfeifenraucher-Zeitung „Radford's News" die Wunschpfeife ihrer Leser entwerfen lassen. Bisher sind über 3000 Leser der Aufforderung gefolgt, einen zweidimensionalen Entwurf bei der Redaktion einzureichen. Eine Reihe möglichst unterschiedlicher „Modelle" wird dann von der „Academie Internationale de la Pipe" ausgewählt und von der Firma System Consult in einen Rechner eingespeist. Er soll daraus die „Wunschpfeife '87" entwickeln, die dann qualifizierte Preifenmacher im französischen Pfeifen-Mekka St. Claude in limitierter Auflage herstellen werden. Eines steht heute schon fest: Die Sammler werden sich um dieses Objekt reißen.



Pfeifen-Design mit Computerhilfe: Hier ein Entwurf von Werner Karl aus Kronach

Handbuch für den IBM-PC

Programmierhandbuch für den IBM-PC. Von Peter Norton. 403 Seiten, Kart. 89 DM. Vieweg Verlag, Braunschweig.
ISBN 3-528-04489-6

Peter Norton skizziert kurz den Aufbau des PCs und des ATs und schildert die Aufgaben einiger Schlüsselbausteine wie des Interrupt-Controllers 8259, des CRT-Controllers 6845 und des DMA-Controllers 8237A. Die Grundlagen der Programmierung des 8088 werden sehr kurz dargestellt. Die Interrupts des PCs werden nicht nur beschrieben, sondern es wird mit Beispielen gezeigt, wie die Interruptvektoren geändert werden können. Detailliert dargestellt wird der Kontrollinformationsbereich des BIOS, dás ist der Speicherbereich zwischen 400...500H. Die Kapitel DOS-Grundlagen, DOS-Interrupts, traditionelle und neue DOS-Funktionen schildern auf über 90 Seiten fast alles, was man schon immer über DOS wissen wollte. In einer Zusammenfassung aller DOS-Routinen wird angegeben welche, Register zur Ausführung der Funktion benötigt werden. Mich hat bei der Beschreibung der DOS-Funktionen etwas irritiert, daß im Text fast immer die dezimale Schreibweise der DOS-Funktionen verwendet wurde. Eine Frage, die jeden Programmierer bewegt, ist die, wie Module in höheren Programmiersprachen mit Assemblerroutinen verbunden werden. Der Autor gibt Beispiele für interpretierendes und kompiliertes Basic, Microsoft-Pascal und C (Lattice und Microsoft). Nach Turbo-Pascal sucht man leider vergebens.
Peter Nortons Buch darf zu Recht als umfassendes Nachschlagewerk für den IBM-PC bezeichnet werden. St

Ihr Epson-Drucker

David A. Kater, Richard L. Kater, Ihr Epson-Drucker, 216 Seiten, kart., Verlag McGraw-Hill, 48 DM.
ISBN 3-89028-050-1

Das Buch „Ihr Epson-Drucker" ist eine Ergänzung zum Handbuch der Epson-Drucker der MX-,RX-,FX- und LQ-Serie und es richtet sich vornehmlich an Anfänger, die das erste Mal mit einem Matrix-Drucker zu tun haben. Nach einem kurzen Abriß der Entwicklungsgeschichte der Druckerfamilie von Epson wird die Arbeitsweise eines Nadeldruckers anschaulich erläutert. Die Autoren führen in den Gebrauch der Fluchtsymbole zur Steuerung der verschiedenenen Zeichenvarianten, des Papiervorschubs und der Schalter-Voreinstellungen ein. Dabei läßt sich nicht verhindern, auch Informationen aus dem Handbuch noch einmal aufzuführen. Die Autoren beschränken sich hier auf das Notwendigste. In den folgenden Kapiteln werden die verschiedenen Punktgrafik-Modi behandelt und gezeigt, wie man eigene Zeichensätze definiert. Weitere Beispiele zeigen das Entwerfen von Zeichen, die sich über mehrere Zeilen erstrekken. Danach wird die Anpassung einiger Druckfunktionen an Wordstar und Lotus 1-2-3 vorgeführt. In den letzten beiden Kapiteln zeigen die Autoren, wie man in Basicprogrammen die Epson-Drucker steuert und wie man selbstentworfene Formulare zu Papier bringt. Das Buch ist eine gute Einstiegshilfe und bietet dem Anfänger brauchbare Tips. *JPL*

Die Technik der Mikrocomputer

Handbuch der digitalen Elektronik, Teil 4. Von Jan Hendrik Jansen. 359 Seiten, 116 Abbildungen, Lwstr.-gebunden mit Schutzumschlag, 78 DM. Franzis-Verlag, München, 1987.
ISBN 3-7723-7951-6

Das Buch trägt den Untertitel: Die Kombination der Teilschaltungen zu einem programmierbaren, datenverarbeitenden System. Der Aufbau und die Funktion eines 8-Bit-Mikrocomputers werden ausführlich und anschaulich behandelt. Jan Hendrik Jansen stellt die Lösung des Problems der Systemanalyse anhand von treffenden Beispielen aus dem täglichen Leben dar. Die Architektur eines Mikrocomputers wird mit einem 8085A-Prozessor beschrieben. Der Befehlssatz dieses Prozessors sowie der Aufbau eines Mikroprozessor-Entwicklungssystems mit den Ein-Ausgabebausteinen 8255A und 8251A wird ausführlich erläutert. Auch wie ein EPROM-Programmiergerät für das EPROM 2708 gebaut wird, erfährt der Leser. Auf Sprachsynthese, Spracherkennung und die Programmiersprache Basic wird detailiert eingegangen. Mit einem Blick auf neue Entwicklungen auf dem Gebiet der Mikroprozessoren endet das Buch, das durch Anschlußbilder der behandelten ICs sowie übersichtliche Grafiken, Schaltpläne und Tabellen ergänzt ist. Es eignet sich sowohl als Nachschlagewerk für den Fachmann als auch als Lehrwerk für den branchenfremden Ingenieur, der sich in die Organisation und den Aufbau eines Mikroprozessorsystems einarbeiten möchte. Wegen der präzisen und ausführlichen Darstellung und der Aufmachung des Buches ist es sein Geld wert. *Volker Bollig*

Die Programmiersprache Lisp

Einführung in die Sprache der künstlichen Intelligenz. Von Dr. Georg Schoffa. 184 Seiten. Lwstr.-kart. 48 DM. Franzis-Verlag, München, 1987.
ISBN 3-7723-8361-0

Die Entwicklung der künstlichen Intelligenz hat das Interesse für einige höhere Sprachen gefördert, von denen Lisp neben Prolog und Ada eine der bekanntesten ist. Das Buch von Dr. Schoffa beschäftigt sich mit CommonmuLisp-86, einem Lisp-Dialekt, der nach der letzten IJCAI-Konferenz ein inoffizieller Standard ist. Auf PCs hat dieser Dialekt aber eine große Verbreitung gefunden. Alle wichtigen Strukturen der Sprache Lisp werden in dieser Einführung ausführlich und eingehend behandelt. Mit zahlreichen Beispielen wird der praxisorientierte Anspruch des Buches immer wieder unterstrichen. Von einem Abriß über die Entwicklung der Sprache Lisp über eine Aufstellung verfügbarer Lisp-Interpreter für den IBM-PC bis hin zum Vergleich von unterschiedlichen Programmiertechniken spannt sich der Bereich, der durch das Buch abgedeckt ist. Dabei werden alle Aspekte berücksichtigt, die von Programmierern bei ihrer Arbeit zu beachten sind. Nach der Lektüre des Buches ist der Leser in der Lage, eigene Problemlösungen in Lisp zu formulieren beziehungsweise zu programmieren. Dr. Schoffa führt den Leser systematisch in die lisp-spezifischen Strukturen ein und macht ihn mit der Denkweise der künstlichen Intelligenz vertraut. *rs*

Rainer Hoffmann

Wenn die neue Harddisk nicht läuft

Tips für die Installation von Festplatten auf PCs

In letzter Zeit sind die Preise von Festplatten so stark gesunken, daß sich immer mehr PC-Besitzer zum Kauf entschließen. Und wer erst einmal damit gearbeitet hat, möchte auf die damit verbundene Bequemlichkeit nicht mehr verzichten. Leider gibt es anfangs oft Ärger, denn Festplatten werden nicht betriebsbereit ausgeliefert. Dieser Beitrag hilft Ihnen vielleicht weiter.

Die günstigsten Festplatten haben einen Schrittmotor und dadurch mittlere Zugriffszeiten (Average Access Time) von 65 bis 85 ms. Dieser Wert ist ein besonderes Qualitätsmerkmal. Kapazitäten zwischen 10 und 30 MByte sind üblich. Meistens fehlt hier jede Sicherheit für die Daten des Benutzers.
Es gibt eine Reihe solcher Sicherheitseinrichtungen. Als erstes den Head-Lifter, der die Köpfe bei Spannungsabfall oder bei zu hohen Beschleunigungskräften von den Platten abhebt. Als zweites die Parkspur-Automatik, die die Köpfe bei Spannungsausfall noch rechtzeitig auf die Parkspur bewegt.
Bei sogenannten Drivecards, also Laufwerken, die auf einer Karte untergebracht sind, müssen die Verlustleistung des gesamten Moduls und die Einbauposition beachtet werden, da die Wärmestrahlung auch auf benachbarte Karten Einfluß nimmt. Es hat wenig Sinn, eine Drivecard mit niedriger Zugriffszeit (bei hoher Verlustleistung) einzusetzen und sich dabei durch Erhöhen der Innentemperatur Rechnerabstürze einzuhandeln. Festplatten mit Servokopf sind wohl das Beste, das es auf dem Markt zur Zeit gibt. Die Zugriffszeiten liegen bei 23...55 ms, die Kapazität zwischen 30...160 MByte.

Die Platten eines Laufwerks dehnen sich bei Erwärmung aus, auch wenn die Technik durch spezielle Tricks diese Dehnung minimiert hat. In einem Laufwerk mit Schrittmotor-Kopfantrieb, das zwischen 11 und 25 W verbraucht, verschiebt sich die Spurlage nach längerem Betrieb gegenüber dem Kopf etwas. Deshalb legt man die Abstände der Spuren und ihre Breite größer aus, damit keine Schreib-/Lesefehler auftreten können. Wenn jedoch ein Servokopf, der durch eine Art Linearmotor angetrieben wird, die anderen fest verbundenen Köpfe auch bei Ausdehnung nachführt, kann man die Abstände und Breiten geringer halten. Es passen somit auch mehr Spuren auf die Plattenoberfläche. Natürlich gehört auch eine aufwendigere Elektronik für die Datentreiber (Rauschabstand) dazu.
Auch bei den Controllern gibt es Unterschiede. Benchmark-Tests an einem 20-MByte-Drive des Typs TM262 mit einer mittleren Zugriffszeit von 85 ms (laut Hersteller) ergaben mit Omti-Controller 82,7 ms und mit WD-Controller (Western Digital) 126,5 ms mittlere Zugriffszeit. Nachdem nun WD den Controller auf halbe Baugröße geändert hat (Made in Ireland), sollte man meinen, auch die Bios-Routinen seien überarbeitet. Weit gefehlt, es ist die gleiche Schaltung, nur mit SMD-Bauteilen.
Viel Ruhe und Geduld braucht man, um eine Festplatte im PC oder AT zu installieren. Nachfolgend erfahren Sie, wie man dabei am besten vorgeht.

Tabelle 1. Harddisk-Kennwerte des AT-Bios

Typ	Cyl.	Heads	WPC	Landing Zone
1	306	4	128	305
2	615	4	300	615
3	615	6	300	615
4	940	8	512	940
5	940	6	512	940
6	615	4	–	615
7	462	8	256	511
8	733	5	–	733
9	900	15	–	901
10	820	3	–	820
11	855	5	–	855
12	855	7	–	855
13	306	8	128	319
14	733	7	–	733
15	Reserviert			

Hardware-Voreinstellung

Das Laufwerk (Drive) wird als erstes auf die richtige Jumper-Stellung hin untersucht. Der Jumper für das logische Laufwerk C: muß bei IBM-PCs auf Laufwerk 2 stehen (wenn man von 1 zu zählen beginnt), da am Ende (Laufwerk C:) des breiten Kabels (Steuerleitungen) ein paar Adern, nämlich die Select-Leitungen, verdreht sind. Einige Hersteller kompatibler Rechner haben das kopiert. Dadurch wird der Select von DS0 auf DS1 gelegt, das Laufwerk C: ist also mit der zweiten Select-Leitung verbunden. Daraus folgt: Alle angeschlossenen Laufwerke haben den Select-Jumper auf der zweiten Position (DS2).
Mit der zweiten Einstellung wird es komplizierter: Das Bios für die Harddisk-Routinen befindet sich immer auf dem Controller. In jedem Bios-EPROM ist eine Tabelle, die passend zur Größe des Laufwerks selektiert werden muß. Beim WD-Controller ist das SW1, Brücke 1...4 (3 und 4 = Select 0, 1 und 2 = Select 1). Das ergibt vier Möglichkeiten pro Laufwerk. Man kann hier also zwei unterschiedlich große Festplatten-Laufwerke anschließen.

Tabelle 2. Kennwerte verschiedener Harddisk-Laufwerke

Modell	Cyl.	Heads	WPC	Landing	Total Bytes	Av. Access
Atasi						
3064	645	7	323	644	39 298 560	33
3051	704	7	352	703	42 893 312	
3051+	733	7	368	732	44 660 224	
3085	1024	8	–	1023	71 303 168	29
CDC						
9420x-51	989	5	0	989	43 041 280	28
9415x-21	697	3	0	697	18 200 064	
9415x-36	697	5	0	697	30 333 440	
9415x-48	925	5	0	925	40 256 000	28
9415x-57	925	6	0	925	48 307 200	
9415x-67	925	7	0	925	56 358 400	28
9415x-77	925	8	0	925	64 409 600	
9415x-86	925	9	0	925	72 460 800	28
CMI						
CM-6426	615	4	300	615	21 411 840	
CM-6640	615	6	300	615	32 117 760	
Fujitsu						
M2241AS	754	3	375	754	19 688 448	33
M2242AS	754	7	375	754	45 939 712	33
M2243AS	754	11	375	754	72 190 976	33
Hitachi						
DK511-3	699	5	300	699	30 420 480	30
DK511-5	699	7	300	699	42 588 672	30
DK511-8	823	10	400	822	71 633 920	25
IBM						
20 MByte (2)	615	4	128	305	21 411 840	
20 MByte (13)	306	8	128	319	21 307 392	
30 MByte (22)	733	5	300	733	31 900 160	
Lapine						
Titan 20	615	4	–	615	21 411 840	65
Titan 20 RLL	615	4	–	615	32 747 520	65
Maxtor						
XT1085	1024	8	–	1023	71 303 168	28
XT1105	918	11	–	1023	87 892 992	27
XT1140	918	15	–	1023	119 854 080	27
XT2085–	1024	7	–	1023	62 390 272	
XT2140–	1024	11	–	1023	98 041 856	
XT2190–	1024	15	–	1023	133 693 440	
XT2085	1224	7	–	1223	74 575 872	30
XT2140	1224	11	–	1223	117 190 440	30
XT2190	1224	15	–	1223	159 805 440	30
Micropolis						
1302	830	3	–	829	21 672 960	
1303	830	5	–	829	36 121 600	
1304	830	6	–	829	43 345 920	
13x3	1024	4	–	1023	35 651 584	
13x3A	1024	5	–	1023	44 564 480	28
13x4	1024	6	–	1023	53 477 376	
13x4A	1024	7	–	1023	62 390 272	
13x5	1024	8	–	1023	71 303 168	28
Microscience						
HH-325	615	4	–	615	21 411 840	
HH-725	615	4	–	615	21 411 840	
HH-1050	1024	5	–	1023	44 564 480	
Miniscribe						
6032	1024	3	–	1023	26 738 688	
6053	1024	5	–	1023	44 564 480	
6074	1024	7	–	1023	62 390 272	
6085	1024	8	–	1023	71 303 168	
3425	615	4	300	615	21 411 840	
3412	615	4	–	615	21 411 840	
Mitsubishi						
MR522	612	4	300	612	21 307 392	
MR533	971	3	–	971	25 354 752	
MR535	971	5	–	971	42 257 920	

Modell	Cyl.	Heads	WPC	Landing	Total Bytes	Av. Access
NEC						
D5124	309	4	–	664	10 758 144	
D5126	612	4	–	644	21 307 392	
D5146	615	8	–	664	42 823 680	
Newbury						
NDR320	615	4	–	615	21 411 840	
NDR340	615	8	–	615	42 823 680	
NDR1085	1024	8	–	1023	71 303 168	
NDR1105	918	11	–	1023	87 892 992	
NDR1140	918	15	–	1023	119 854 080	
NDR2190–	1024	15	–	1023	133 693 440	
NDR2190	1224	15	–	1223	159 805 440	
Priam/Vertex						
Bull						
V130	987	3	–	987	25 772 544	
V150	987	5	–	987	42 954 240	
V170	987	7	–	987	60 135 936	
V185–	1024	7		1023	62 390 272	
519–	1024	15	–	1023	133 693 440	
V185	1166	7	–	1165	71 042 048	
519	1224	15	–	1223	159 805 440	
Quantum						
Q520	512	4	256	512	17 825 792	
Q530	512	6	256	512	26 738 688	
Q540	512	8	256	512	35 651 584	
Rodime						
203	321	6	132	321	16 763 904	
204	321	8	132	321	22 351 872	55
202E	640	4	0	640	22 282 240	
203E	640	6	0	640	33 423 360	55
204E	640	8	0	640	44 564 480	55
652	612	4	?	?	21 307 392	65
SCSI						
752	612	4	?	?	21 307 392	65
SCSI						
Seagate						
ST213	615	2	300	615	10 705 920	
ST225	615	4	300	615	21 411 840	65
ST238	615	4	300	615	32 747 520	65
RLL						
ST412	306	4	128	305	10 653 696	
ST251	820	6	–	820	42 823 680	40
ST277	820	6	–	820	42 832 680	
ST4026	615	4	300	615	21 411 840	40
ST4038	733	5	300	733	31 900 160	40
ST4038M	733	5	–	977	31 900 160	
ST4051	977	5	–	977	42 519 040	40
ST4096	1024	9	–	1023	80 216 064	28
Tandon						
TM262	615	4	–	615	21 411 840	85
TM755	981	5	–	981	42 693 120	
TM362	615	4	–	615	21 411 840	
TM703	695	5	–	695	30 246 400	
TM705	962	5	–	962	41 866 240	
TM702AT	615	4	–	615	21 411 840	
TM703AT	733	5	–	733	31 900 160	
Toshiba						
MK-53F	830	5	512	830	36 121 600	
MK-54F	830	7	512	830	50 570 240	
MK-56F	830	10	512	830	72 243 200	
Tulin						
TL226	640	4	–	640	22 282 240	
TL326	640	4	–	640	22 282 240	
TL240	640	6	–	640	33 423 360	
TL340	640	6	–	640	33 423 360	
TL238	640	4	–	640	22 282 240	
TL258	640	6	–	640	33 423 360	

Beim Omti-Controller (5510/20/27) ist die Größe von der Bios-Version im EPROM abhängig. Auch der RLL-Controller hat eine andere Tabelle als der normale, auch gilt hier bei Anschluß von zwei Festplatten die Tabelle für beide Laufwerke. Die Brücken sind W1 bis W4, das ergibt 16 Möglichkeiten. Es können maximal zwei gleich große Laufwerke angeschlossen werden.
Danach werden die Kabel aufgesteckt. Zuvor sollte man die Kontakte reinigen, weil eine Festplatte erhebliche Wärme entwickeln kann und Fingerabdrücke dadurch sehr schnell oxidieren. Pin 1 am Laufwerk (die Markierung) soll immer nahe zum Verdrehungsschlitz aufgesteckt werden.
Auf dem Controller muß man sich die Bezeichnung für den ersten Pin suchen. Sollte trotzdem ein Kabel irgendwie verdreht sein, ist das nicht so schlimm, da kurzzeitig keine Schäden auftreten können.
Wenn das Steuerbus-Kabel (breites Kabel) verdreht ist, wird die Select-Lampe sofort beim Einschalten aufleuchten und nicht mehr verlöschen. Diese Aussage gilt jedoch nur für PCs/XTs. Beim AT sieht man das an der Frontlampe.
Wenn das Datenbus-Kabel (schmales Kabel) verdreht ist, wird das Laufwerk überhaupt nicht „ready", weil Pin 1 eine Steuerleitung für angeschlossene Laufwerke ist und dadurch der Controller physikalisch kein Laufwerk erkennt.
Nun wird das Laufwerk an die Stromversorgung angeschlossen und der Controller in den Erweiterungssteckplatz gesteckt. Da die Festplatte noch lose auf dem Gerät liegt, legt man ein dünnes Buch oder etwas Pappe darunter. Bei defektem Gerät erspart man sich dadurch den erneuten Ausbau!
Festplatten können nur in drei Lagen eingebaut werden, seitlich hochkant (auf eine der beiden Seiten) oder flach mit der Elektronik nach unten. In Ausnahmefällen ist die Elektronik auch oben. Der Einbau auf dem Kopf liegend ist für die Spindellager nicht vorgesehen und führt zu Lagerschäden. Disketten-Laufwerke sollte man so einbauen, daß sie nicht unmittelbar „neben" der Harddisk-Elektronik liegen, da sonst Störungen aus der Elektronik in den Kopf der Floppy einstrahlen und starke Schreib-/Lesefehler verursachen (im allgemeinen also über der Festplatte).

Nach Einschalten des Rechners sollte der Speicher hochzählen und danach den Fehler „1701 F1 Taste drücken" auf dem Bildschirm anzeigen, wenn die Festplatte zum erstenmal angeschlossen wird. Das kann rund zehn Minuten dauern (Geduld ist alles). Nach dem Drükken der besagten Taste bootet das System, und es geht ganz normal weiter.

Wenn man nicht über 32 MByte hinaus will, dann braucht man nur noch die Spezifikation der Festplatte und kann formatieren. MS-DOS verwaltet nur Platten bis zu dieser Größe. Bei größeren Kapazitäten empfiehlt sich spezielle Software.

Software-Größen

Diese Angaben braucht man nur, wenn mit externer Software oder der im Bios befindlichen Extern-Input-Version formatiert wird, dazu helfen auch die *Tabellen 1 und 2*.

Wie viele Zylinder hat das Laufwerk?

Wenn Sie einen Exoten formatieren und nur annähernde Informationen über die Zylinderzahl haben, nehmen Sie die am nächsten vermutete in der Tabelle 2 stehende Zylinderzahl mit der gleichen Kopfanzahl, dabei geht zwar etwas Speicher verloren, das Laufwerk läßt sich aber ohne Schwierigkeiten formatieren.

Wieviel Köpfe?
Eine falsche Angabe kann zum Absturz des Systems führen.

Bei welchem Zylinder beginnt der Reduced Write Current?

Wenn nicht bekannt, dann bei 2/3 oder letzter Zylinder.

Bei welchem Zylinder beginnt die Write Precompensation?

Auch hier den letzen Zylinder angeben.

Wie groß ist die Max Correctable Error Burst Length?

Beim PC und AT immer 11.

Wie groß ist das CCB Option Byte?

Beim PC und AT immer 5.

Wie groß ist der Interleave-Faktor?

Das ist die Differenz der logischen Nummern zweier aufeinanderfolgender Sektoren in einer Spur. Im PC/XT wird meistens 6 angegeben, im AT 3. Bei schnellen Controllern mit „Bufferd Track" kann man auch beim PC/XT einen Interleave-Faktor von 3 angeben (z.B. Omti).

Formatprogramm von Laufwerk A: starten

Die einfachste Art, ein Laufwerk zu formatieren, ist es, den Debugger aufzurufen und folgendes einzugeben:

G=C800:5 (return) beim WD- und DTC-Controller
G=C800:6 (return) beim Omti-Controller

Damit beginnt eine Low-Level-Format-Routine.

Jetzt sind je nach Programm (Autoconfig oder Intern Input) zwei Möglichkeiten offen:

1. Die Software meldet sich mit der Abfrage nach Drive 0 oder 1 und will eine Interleave-Faktor Eingabe. Nach einem (Y)es startet der Formatvorgang, wenn eine feste Vorgabe für die Drive-Konfiguration durch Jumper eingestellt wurde und die Controller-Software keine manuelle Eingabe zuläßt.

2. Die Software fragt nach Zylinder, Total Number of Heads, Reduced Write Current, Write Precompensation, Max Correctable Error Burst Length und CCB Option Byte. Dieses wird in der abgefragten Form eingegeben und mit Return bestätigt. Danach folgt die Abfrage über die virtuelle Aufteilung des Plattenspeichers. Wenn man aus einem 20-MByte-Drive ein Laufwerk C: und D: machen will, muß man nach der Bestätigung mit (Y)es zwei Werte für die Zylinderzahl pro logischem Laufwerk (Volume) eingeben. Diese Werte dürfen die Gesamtkapazität nicht überschreiten. Danach folgt auch hier die Formatierung. Dieser Vorgang startet, wenn die Autoconfig-Option in der Bios-Software vorhanden ist.

Bei mehr als einer Partition (logischem Laufwerk) sollte jedoch ein Software-Format-Programm zu Hilfe genommen werden (Speedstro, Golden Bow, Mountain, OmtiDisk). Mit einigen dieser Programme kann man auch eine Partition pro logischem Laufwerk bis maximal 140 MByte erstellen.
Formatiert wird mit 17 Sektoren pro Spur und 512 Byte pro Sektor. Dabei wird, wenn das Laufwerk einen Servokopf hat, auch die Servo-Oberfläche mitformatiert. Bei RLL-Konfigurationen ändert sich die Sektorenzahl auf 26 (Omti 5527).
Spätestens jetzt können Fehlermeldungen auftreten, und die Formatierungs-Routine bricht ab. Einige Ursachen sind dann: eine defekte Harddisk, falsche Jumper-Stellung oder falsche Angaben.

Ist IBM zu sich selbst noch kompatibel?

Probleme mit Harddisks bei den neuen XT-Typen

Das Arbeiten mit einer Harddisk ist eine schöne Sache. Immer mehr PC-Benutzer kommen zu dieser Erkenntnis. Ich selbst bestreite meinen Lebensunterhalt damit, Harddisks an PCs anzupassen (wenigstens zum Teil), und weil viele meiner Feunde sich in letzter Zeit Plattenspeicher angeschafft haben, muß ich immer öfter den „Retter in höchster Not" spielen, weil irgend etwas nicht klappt.

Im allgemeinen lassen sich solche Hilferufe schnell erledigen, schließlich hat man sich im Lauf der Zeit einiges an Routine und Wissen angeeignet. Aber manchmal verschlägt es einem selbst dort die Sprache, wo man glaubt, alle Geheimnisse entschlüsselt zu haben. So geschehen kurz vor Weihnachten, als mich ein Bekannter um einen kurzen Besuch bat, weil sich meherere vorher getestete Harddisks nicht formatieren ließen. Aus dem Besuch wurden drei Tage Arbeit, die man sich hätte sparen können.

Der Computer war ein IBM-PC aus der neuen XT-Serie mit der Bezeichnung XT-SFD. Festplatte und Controller (beides nicht von IBM) ließen sich mit keinem Trick dazu bewegen, auch nur ihre Existenz im Rechner zuzugeben. Die Boot-Routine wurde überhaupt nicht angesprochen. Irgend etwas mußte sich anders verhalten als früher.

Zunächst war da eine neue Tastatur. Alles, was man am alten Platz seit zwei bis drei Jahren gewohnt ist, befindet sich auf ihr ganz woanders – von wegen „kompatibel". Dafür gibt es aber ein gutes Dutzend Tasten mehr. Wir tauschten als erstes die Tastatur gegen eine altgewohnte aus. Nun war der Rechner endlich bereit, seinen Bootstrap auszuführen, und das Laufwerk ließ sich formatieren.

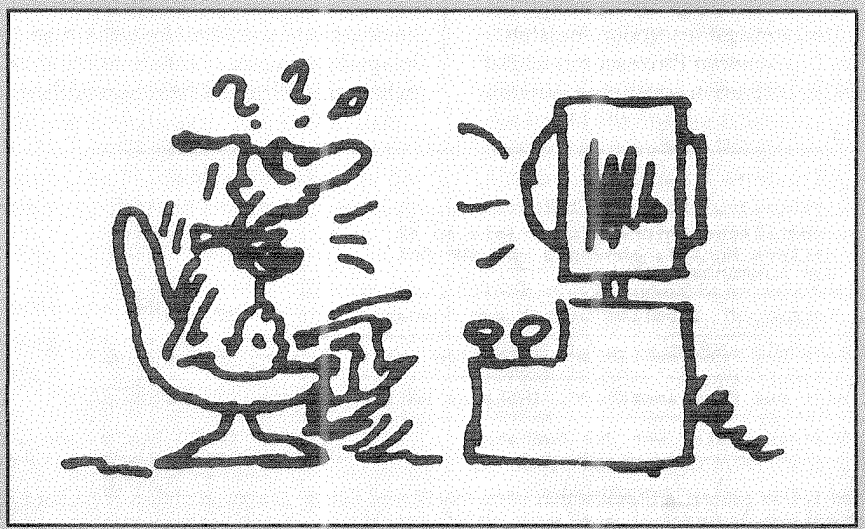
Nach drei Stunden Test wollte ich mich gerade auf den Weg nach Hause begeben, als (wie üblich) einer der Kollegen meines Bekannten am Rechner spielte, und das Laufwerk sich wieder verabschiedete. Der Rechner blieb wieder in der Boot-Routine hängen. Wir tauschten dann auch noch den Controller aus, aber der Erfolg blieb aus. Damit war der erste Tag um.

Am nächsten Tag fiel mir das neue BIOS mit der Bezeichnung 62X0854 auf, das in zwei EPROMs des Typs 27256 untegebracht ist. Auf der XT-Platine sah man die zusätzlich eingefügten Leitungen für die Pins 27 und 28. Das war mir neu, denn IBM hatte bis dahin nichts über ein neues BIOS verlauten lassen, obwohl diese Firma sonst mit Information nicht geizt. Versuchsweise setzten wir (nach einem Anruf bei IBM) das alte BIOS ein, und siehe da: Es funktionierte.

Das Problem an der Sache war jetzt nur, daß der Bekannte einen IBM-XT-SFD mit 20-MByte-Harddisk haben wollte, und nicht – wie aus unseren Abspeckversuchen entstanden – die alte XT-Version. Wir probierten deshalb noch mit anderen DOS-Versionen, Controllern und Laufwerken herum – alles ohne Erfolg. Damit war der zweite Tag um.

Am dritten Tag erfuhr ich über Umwege (nicht von IBM selbst) von zwei IBM-Disketten zur Tastatur- und DOS-Anpassung, die – wie weitere Nachforschungen ergaben – manchen, aber nicht allen Rechnern beigelegt sind. Damit gelang es endlich, den Rechner im Originalzustand mit Fremdlaufwerk und DOS 3.1 zum Laufen zu bringen.

Ich frage mich, wie jemand mit der Nachrüstung eines solchen Rechners fertig werden soll, wenn er eine andere DOS-Version besitzt oder von den Disketten nichts weiß? Den Geräten von IBM liegen viele Beschreibungen bei, unter anderem, in welche Richtung man eine Schraube drehen muß (mit genauer Zeichnung), aber hier „vergißt" man einfach wesentliche Informationen. Wo ist der Hinweis auf das neue BIOS? Es muß erhebliche Unterschiede zu der bei uns als „Industrienorm" bekannten Version geben. Ist der neue XT nicht mehr kompatibel zu IBM-PC-DOS? Mich würde interessieren, was „Mama Blue" dazu sagt. *Rainer Hoffmann*



Werner Stehling

Interrupts bei Prozessoren 680XX

Teil 2: Stack, Hardware und Programmierung

Nachdem im ersten Teil geschildert wurde, was Ausnahmen sind und wie der Prozessor nach Eintreten einer Ausnahmebedingung die Adresse der zuständigen Routine findet, geht es nun ganz ins Detail: Die Unterschiede der verschiedenen 680XX-CPUs, die Anforderungen an die Hardware und hilfreiche Regeln für die Programmierung von Interrupt-Routinen sind Inhalt dieses zweiten und letzten Teils unserer Einführung.

Auf die eine oder andere Art hat die CPU also nach Auftreten der Ausnahme eine Vektornummer ergattern können. Durch Multiplikation mit 4 erhält sie den Offset, d. h. die relative Adresse, in der Tabelle der Ausnahmevektoren. Sie weiß damit, wo die Routine beginnt, mit der sie auf die Unterbrechung des Programmes reagieren soll.

Der wichtigste Grundsatz bei der Ausnahmeverarbeitung ist, daß das unterbrochene Programm nichts davon merken darf. Es soll nach Beendigung der Ausnahmeverarbeitung so weiterlaufen, als ob diese überhaupt nicht stattgefunden hätte, wenigstens in den meisten Fällen. Die neueren Prozessoren 68010 und 68020 erlauben sogar eine Wiederholung des letzten Befehls, falls dieser durch einen Busfehler unterbrochen wurde.

Rettungsaktion

Um so intelligent agieren zu können, muß am Ende der Ausnahmeverarbeitung der innere Zustand der CPU genau so sein wie vor Beginn. Das Retten der Arbeitsregister bleibt dem Programmierer überlassen, dagegen gibt es einige interne Informationen, auf die der Anwender keinen Zugriff hat, die aber vom Prozessor vor dem Sprung in die Ausnahmeverarbeitungsroutine auf den Stack gerettet werden. Um nur einige Beispiele zu nennen: Inhalt von Statusregister und Programmzähler vor dem Auftreten der Ausnahme, Status der Funktionscode-Leitungen usw. Je nach Art der Ausnahme müssen mehr oder weniger Informationen aufbewahrt werden, bei einem Busfehler ist es zum Beispiel interessant, ob als letztes auf einen Befehl oder auf einen Operanden zugegriffen wurde; das wiederum ist bei einem normalen Interrupt völlig ohne Belang, da der laufende Befehl in jedem Fall vor der Annahme des Interrupts vollständig abgearbeitet wird.

Die einzige Ausnahme von dieser Regel ist ein externes Reset-Signal. Normalerweise wird es nur beim Einschalten der Stromversorgung oder als allerletzte Notbremse bei chaotischem Verhalten des Mikrocomputers angelegt. In beiden Fällen erfolgt sowieso eine völlige Neubesinnung des Systems auf seine Aufgabe. Es ist also nur logisch, wenn die CPU beim Reset erst gar keine Anstalten unternimmt, irgendwelche internen Informationen zu retten. Wohin sollte sie auch – der Stackpointer wird ja gleich zu Beginn der Ausnahme neu initialisiert! Zwischen den Prozessoren der Familie 68000 bestehen zusätzliche Unterschiede, immer getreu der Devise: Je intelligenter ich bin, desto mehr muß ich mir merken.

Wenden wir also unsere Aufmerksamkeit jetzt den verschiedenen Stackformaten zu. Im ritualisierten Ablauf der Ausnahmeverarbeitung rettet die CPU alles auf den Stack, von dem sie glaubt, daß es während oder nach der Ausnahmeverarbeitung von irgendeinem Nutzen sein könnte.

Verschiedene Stackformate

Wie es sich gehört, liegt der Supervisor-Stack im Supervisor-Datenbereich, was sich auch bei Stackzugriffen im Zustand der Funktionscode-Leitungen widerspiegelt. Es wird der Supervisor-Stackpointer aktiviert, der 68020 verwendet bei Interrupts zusätzlich einen speziellen Interrupt-Stackpointer. Der Stack wächst immer von höheren zu niedrigen Adressen; der kleinste Stackeintrag ist ein Wort (16 Bit = 2 Byte) und der Stackpointer zeigt immer auf den letzten gültigen Stackeintrag.

Je nach Art der Ausnahme rettet der Prozessor eine verschiedene Anzahl von Worten auf den Stack (auch der 68000). Nach Beendigung der Ausnahmeverarbeitung (genauer: beim Ausführen des RTE-Befehls) regeneriert der Prozessor seinen ursprünglichen Zustand, indem

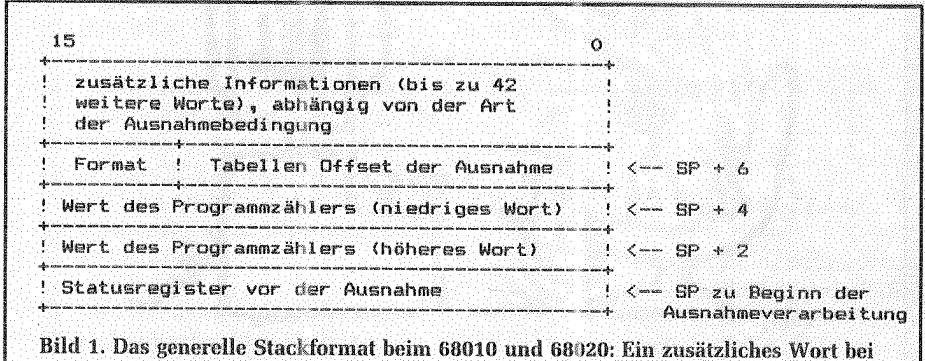


Bild 1. Das generelle Stackformat beim 68010 und 68020: Ein zusätzliches Wort bei SP + 6 enthält die Information, wie viele Worte wirklich auf dem Stack abgelegt wurden („Format") und identifiziert die Ausnahme

er die gleiche Zahl von Worten vom Stack zurückliest, die er zu Beginn dort abgelegt hat. Das Stackformat des 68000 ist grundsätzlich verschieden von dem der beiden anderen Prozessoren, während das Format des 68020 wenigstens aufwärtskompatibel zu dem des 68010 ist.

Beim 68000 können zwei verschiedene Stackformate auftreten, und zwar werden bei Bus- und Adreßfehlern insgesamt sieben Worte gerettet, in allen anderen Fällen nur Programmzähler und Statusregister (zusammen also drei Worte).

Der 68010 kennt ebenfalls nur zwei Formate, allerdings sind es bei Bus- und Adreßfehlern 29 Worte und in allen übrigen Fällen vier Worte. Bereits sechs verschiedene Formate gibt es beim 68020.

Bei 68010 und 68020 enthält das viertunterste Wort auf dem Stack (*Bild 1*) eine Angabe über das verwendete Stackformat (*Tabelle 1*). Die unteren Bit enthalten den Vektor-Offset für die Tabelle der Ausnahmevektoren (was nichts anderes ist, als die mit vier multiplizierte Vektornummer der Ausnahme).

Stackformate bei Bus- und Adreßfehlern

Bild 2 zeigt die Anordnung der Informationen auf dem Stack, wie sie vom 68000 vorgenommen wird. Der Wert des Programmzählers zeigt in die Nähe des Befehls, während dessen Ausführung der Fehler auftrat. Der Wert des Instruktionsregisters gibt den Befehl an, der gerade ausgeführt wurde. Außer der Adresse, die zum Fehler führte, enthält das unterste Wort auf dem Stack noch die Prozessor-Statussignale FC0...FC2. Das dritte Bit im speziellen Statuswort (I/N) ist Null, wenn zum Zeitpunkt des Fehlers gerade eine Ausnahmeverarbeitung der Gruppen 0 oder 1 bearbeitet wird, es ist gesetzt bei „normaler" Programmausführung oder bei einer Ausnahmeverarbeitung eines Befehls aus Gruppe 2. Bit 4 zeigt an, ob der Fehler beim Lesen ($R/\overline{W} = 1$) oder bei einem Schreibzugriff ($R/\overline{W} = 0$) auftrat.

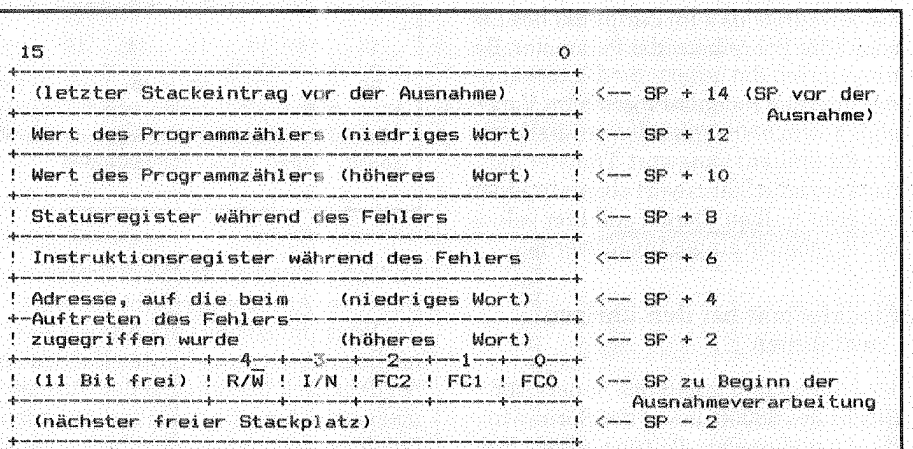


Bild 2. So sieht das lange Stackformat des 68000 bei Bus- und Adreßfehlern aus

Der RTE-Befehl des 68000 holt grundsätzlich nur drei Worte vom Stack zurück, also ACHTUNG:

Bei Bus- oder Adreßfehlern muß die Ausnahmeverarbeitungsroutine dafür sorgen, daß der Stack von allem nicht mehr benötigten Unrat gesäubert wird!

Der 68010 rettet wesentlich mehr Informationen auf den Stack (*Bild 3*) und gibt damit dem Anwender die Möglichkeit, die genaue Fehlerursache lokalisieren und gezielt darauf reagieren zu können. Ob das Leben eines geplagten Programmierers dadurch wirklich einfacher wird, sei jetzt mal dahingestellt. Statusregister, Programmzähler, Befehlsregister und Fehleradresse entsprechen den Informationen beim 68000. Auch hier kann der Wert des Programmzählers auf eine Adresse verweisen, die 1 bis 5 Worte hinter dem eigentlichen Befehl liegt. Die Ursache liegt im „Prefetch"-Mechanismus des Prozessors, der während der Befehlsausführung bereits die nächsten Speicherstellen einliest (Auffüllen der internen Pipeline zur Geschwindigkeitssteigerung).

Im speziellen Statuswort bei SP + 8 sind im Vergleich zum 68000 ebenfalls mehr Informationen enthalten (*Bild 4*). Es gibt den internen Zustand der CPU zum Zeitpunkt der Fehlerbedingung wieder:

- FC0...FC2 ist der bekannte Funktionscode (Status),
- RW ist das Lese-/Schreibflag wie beim 68000,
- BY ist ein Byte-Flag, das anzeigt, ob ein Byte (BY = 1) oder ein Wort (BY = 0) übertragen wurde;
- HB gibt an, ob das niedere (HB = 0) oder das höhere (HB = 1) Byte übertragen wurde,
- RM wird bei einem Read-Modify-Write-Zyklus gesetzt,
- DF ist gesetzt, wenn Daten in den Daten-Eingangspuffer geladen wurden,
- IF ist gesetzt, wenn ein Befehl in den Befehls-Eingangspuffer geladen wurde;
- RR bedeutet ReRun-Flag und muß vor der Rückkehr aus der Ausnahmeverarbeitungsroutine vom Programmierer entsprechend gesetzt werden: RR = 1 bedeutet, daß der Fehler auf Grund der Informationen im speziellen Statuswort durch die Ausnahmeverarbeitung (d. h. per Software) behoben wurde, während RR = 0 dem Prozessor signalisiert, daß er es im Anschluß an den RTE-Befehl noch einmal versuchen soll.

Der 68020 kennt sogar zwei unterschiedliche Stackformate bei Bus- oder Adressfehlern. Wenn der Fehler beim Lesen des Befehlswortes auftritt, muß der Prozessor wesentlich weniger Information

Tabelle 1: Bedeutung des Formatcodes bei 68010 und 68020

Code	68010	68020	Worte	Ausnahme-Bedingung
0000	*	*	4	Normalfall
0001	-	*	4	Interrupt-Dummy-Information
0010	-	*	6	Ausnahme-Befehl (z. B. TRAP)
1000	*	-	29	Busfehler
1001	-	*	10	Coprozessor-Ausnahme
1010	-	*	16	Busfehler bei Befehls-Code
1011	-	*	46	Busfehler bei Parameter

aufbewahren als zum Zeitpunkt des Lesens oder Schreibens der Parameter. Bedingt durch die unterschiedliche Länge der internen Befehls-Pipeline im Vergleich zum 68010 und den zusätzlichen (32 Bit breiten) Langwort-Transfer hat das spezielle Statuswort ebenfalls einen anderen Aufbau. Darauf soll hier jedoch nicht genauer eingegangen werden.

Stackformat bei den übrigen Ausnahmen

Bild 5 zeigt den Inhalt des Supervisor-Stacks des 68000, bevor der Sprung in die Ausnahmeverarbeitung erfolgt. Bei allen Ausnahmen außer bei Bus- oder Adreßfehlern werden nur Statusregister und Programmzähler abgelegt; damit ist die Sache noch bequem zu überschauen. Als Referenz für den Stackpointer (SP) wird wieder sein Wert zu Beginn der Ausnahmeverarbeitungsroutine angegeben.

Der 68010 schreibt ein Wort mehr auf den Stack (*Bild 6*). Die für die CPU interessante Information ist der Formatcode 0000, an dem sie beim Ausführen des RTE-Befehls erkennen kann, daß sie genau vier Worte vom Stack zurückholen darf, um damit die Ausnahmeverarbeitung zu beenden.

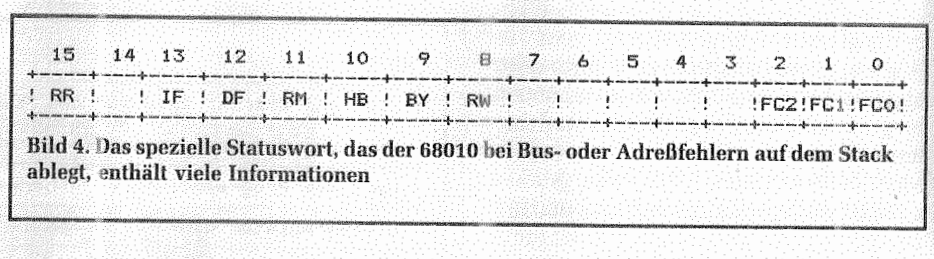


Bild 4. Das spezielle Statuswort, das der 68010 bei Bus- oder Adreßfehlern auf dem Stack ablegt, enthält viele Informationen

Im Unterschied zu den Verhältnissen bei Bus- oder Adreßfehlern gibt der Wert des Programmzählers die Adresse des nächsten nicht ausgeführten Befehls an, bei dem nach Beendigung der Ausnahmeverarbeitung die Arbeit schön ordentlich fortgesetzt wird – oder besser gesagt: werden soll, denn die Ausnahmeverarbeitungsroutine kann natürlich etwas völlig anderes im Sinn haben.

Das gleiche Format (Bild 6) verwendet normalerweise auch der 68020. Speziell bei Interrupts wird diese Information auf den Supervisor-Stack geschrieben. Ist das M-Bit im Statusregister (Bild 1, erster Teil) gesetzt, wurde also zum Zeitpunkt des Interrupts mit dem Supervisor-Stack gearbeitet, so wird das M-Bit gelöscht. Das S-Bit ist zu diesem Zeitpunkt ohnehin gesetzt und die ganze Information wird noch einmal auf den Interrupt-Stack gelegt. Allerdings erhält sie jetzt den Formatcode 0001 (siehe Tabelle 1), damit die CPU später beim RTE-Befehl merkt, daß nicht hier, sondern auf dem Supervisor-Stack die eigentlichen Rückkehr-Daten stehen.
Wenn die Ausnahmeverarbeitung durch einen Befehl herbeigeführt wurde (z. B. CHK oder TRAP), wird zusätzlich noch die Adresse dieses Befehls auf den Stack gerettet (*Bild 7*). Der Wert des Programmzählers zeigt dann auf den nächsten Befehl, mit dem es nach der Ausnahme weitergeht.

Wenn man bedenkt, was die CPU bis jetzt schon alles geleistet hat, ist der Rest nur noch ein Kinderspiel. Der Prozessor nimmt den Vektoroffset der Ausnahme als Index, ein 68010 oder 68020 zählt den Inhalt seines Vektorbasisregisters dazu, und lädt dann den Programmzähler mit der Adresse, die er in der Tabelle 1 (erster Teil) der Ausnahmevektoren findet. So erfolgt ein unbedingter Sprung zur Ausnahmeverarbeitungsroutine, die dann wie ein ganz normales Programm ausgeführt wird.

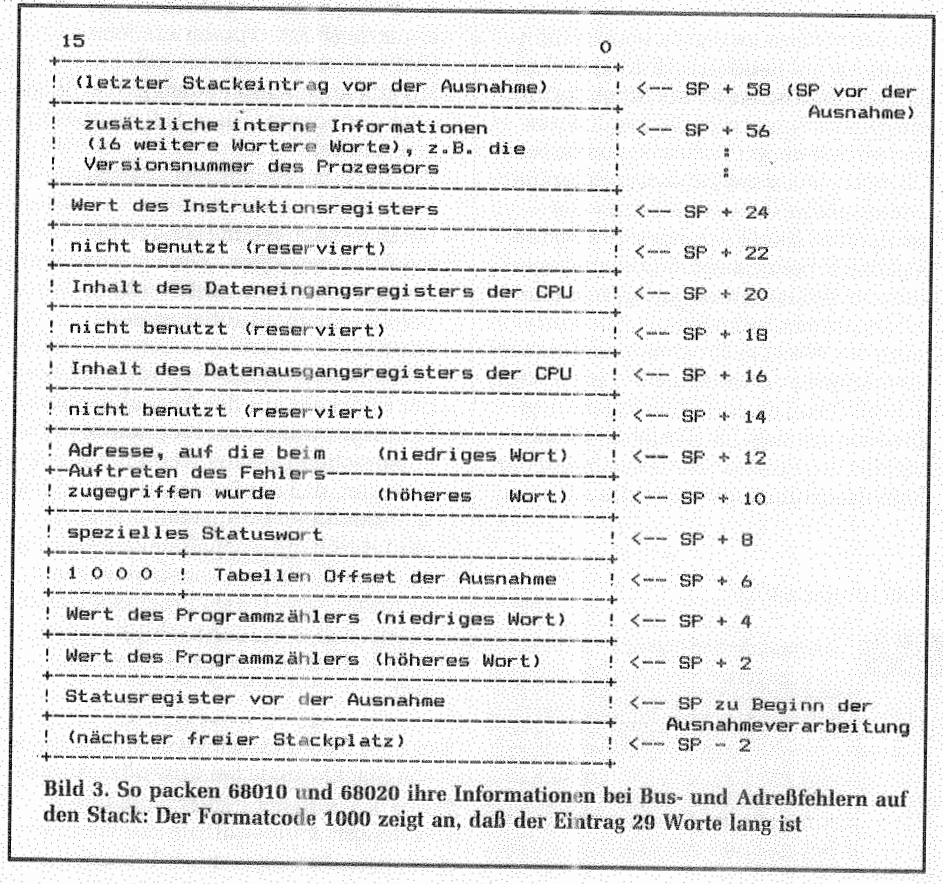


Bild 3. So packen 68010 und 68020 ihre Informationen bei Bus- und Adreßfehlern auf den Stack: Der Formatcode 1000 zeigt an, daß der Eintrag 29 Worte lang ist

Ende der Fahnenstange

Im Normalfall wird die Ausnahmeverarbeitung durch einen RTE-Befehl (= ReTurn from Exception) abgeschlossen. Die Routine hat nach bestem Wissen und Gewissen alle Aufgaben erledigt, für die sie gedacht war; jetzt geht es ans große Stackbereinigen: 68010 und 68020 können am Formatcode erkennen, wieviele Worte sie vorher auf den Stack geschaufelt haben, der 68000 vermutet grundsätzlich nur Statusregister und Programmzähler. Beim Regenerieren des internen Zustandes nimmt es der 68000, wie es gerade kommt, die anderen beiden Prozessoren gehen etwas differenzierter zu Werke:

- Statusregister und Programmzähler werden vom Stack gelesen. Damit ist der 68000 fertig und holt den nächsten Befehl von der Adresse, auf die der Programmzähler zeigt.
- 68010 und 68020 lesen den Formatcode und überprüfen, ob er für den Prozessor überhaupt definiert ist – wenn nicht, dann folgt umgehend eine weitere Ausnahmeverarbeitung über Vek-

tor 14 (Formatfehler). Da der Wert des Stackpointers bis zu diesem Zeitpunkt noch nicht geändert wurde, wird die gesamte zum Fehler führende Stack-Information nicht überschrieben und steht für eine Analyse zur Verfügung.
- Falls der Formatcode das kurze Stackformat (vier Worte) anzeigt, sind auch 68010 und 68020 fertig mit der Ausnahmeverarbeitung und fahren normal im Programm fort.
- Findet der 68020 das kurze Dummy-Format 0001, so lädt er nur das Statusregister vom Stack und führt sofort einen weiteren RTE-Befehl aus. Welcher Stack dabei verwendet wird, bestimmen das M- und S-Bit des gerade gelesenen Statuswortes, es darf also jeder sein.
- Bei einem Busfehler-Format versucht die CPU, das erste Wort auf dem Stack zu lesen. Wenn dabei kein Busfehler auftritt, glaubt der Prozessor (mit einer gewissen Berechtigung), daß die dazwischenliegenden Daten auch alle lesbar sind; im anderen Fall, Sie ahnen es schon, setzt es selbstverständlich eine Busfehler-Ausnahme. Es scheint hier fast, als ob man aus einer Ausnahmeverarbeitung gar nicht mehr herauskommt, wenn man erst einmal damit angefangen hat, aber das scheint eben nur so.
- Die CPU pumpt nun eifrig die nächsten Worte vom Stack und stoppt erst wieder bei einer nicht näher spezifizierten internen Information (SP + 26 in Bild 3, SP + 54 beim 68020). Darin enthalten ist die Versionsnummer des Prozessors, der die Daten auf den Stack geladen hat, und die muß mit der Nummer des lesenden Prozessors übereinstimmen, damit die noch folgenden Stackdaten richtig interpretiert werden. Sollte dem nicht so sein, dann haben wir wieder einen Formatfehler (Sie wissen schon, Vektornummer 14). Das ist z. B. in Multiprozessorsystemen wichtig.
- Wenn wir wirklich bis hierher gekommen sind, ohne eine weitere Ausnahme provoziert zu haben, werden schnell noch die restlichen Daten vom Stack geholt und erst dann wird der Inhalt des Supervisor-Stackpointers korrigiert – bis zu diesem Zeitpunkt hätten ja immer noch Busfehler auftreten können mit der ganzen schönen dazugehörigen Ausnahmeverarbeitung und so fort.
- Jetzt sind wir endgültig fertig.

Soviel zum Normalfall. Die zweite Möglichkeit, eine Ausnahmeverarbeitung zu beenden, funktioniert völlig ohne RTE-Befehl. Dann aber ist der Programmierer der Ausnahmeroutine dafür verantwortlich, daß der Stack bereinigt und die CPU in einen gebrauchsfähigen Zustand versetzt wird. Das bedeutet konkret: Stackpointer um die entsprechende Anzahl Worte korrigieren (wenn Sie jetzt nicht wissen, wie viele, dann setzen Sie einmal beim Würfeln aus und gehen drei Kapitel zurück), das Statusregister mit einem vernünftigen Wert laden und anschließend normal irgendwo im Programmspeicher fortfahren. Die Konsequenz ist natürlich, daß das ursprünglich unterbrochene Programm nicht weitergeführt wird, aber das ist ja vielleicht auch beabsichtigt – so ungefähr nämlich funktioniert Multitasking.

```
 15                                                  0
+-----------------------------------------------------+
! (letzter Stackeintrag vor der Ausnahme)            ! <-- SP + 6  (SP vor der
+-----------------------------------------------------+                Ausnahme)
! Wert des Programmzählers (niedriges Wort)          ! <-- SP + 4
+-----------------------------------------------------+
! Wert des Programmzählers (höheres Wort)            ! <-- SP + 2
+-----------------------------------------------------+
! Statusregister vor der Ausnahme                    ! <-- SP zu Beginn der
+-----------------------------------------------------+     Ausnahmeverarbeitung
! (nächster freier Stackplatz)                       ! <-- SP - 2
+-----------------------------------------------------+
```

Bild 5. Bei allen Ausnahmeverarbeitungen, außer Bus- und Adreßfehlern, verwendet der 68000 dieses kurze Stackformat

Hardware-Voraussetzungen

Wenn man den prinzipiellen Aufbau eines Mikrocomputers betrachtet, können fünf verschiedene Funktionsgruppen unterschieden werden:

(1) Das Gehirn des Ganzen besteht aus der CPU und eventuell vorhandenen Coprozessoren (Arithmetikeinheit, DMA-Controller).

(2) Um das Rechnersystem überhaupt starten zu können, muß ein nichtflüchtiger Speicher vorhanden sein, z. B. ein EPROM oder ein batteriegepuffertes RAM. Häufig werden darin zusätzlich ein Monitor, ein Betriebssystem (komplett oder teilweise) und ein Hochspracheninterpreter (nein, ich sage nicht, welcher!) abgelegt.

(3) Bei Kleinstanwendungen könnte man sich vorstellen, allein mit den CPU-Registern auszukommen. Aber normalerweise wird ein Schreib-Lese-Speicher für irgendwelche Variable und den (System-)Stack vorhanden sein.

(4) Damit der Rechner überhaupt einen Sinn bekommt, muß er irgendwie mit der Außenwelt in Verbindung treten können, etwa über eine Tastatur-Schnittstelle, den Disk-Controller usw.

(5) Das Knochengerüst, das die ganze Chose zusammenhält, wird in den allermeisten Fällen äußerst stiefmütterlich behandelt; es gehört schon riesiges Glück dazu, wenn in offiziellen Verlautbarungen zu einem Computer etwas über Adreß-Decodierung, Power-On-Schaltung oder Interruptbehandlung enthalten ist.

Betrachten wir jetzt einmal den Aufbau unseres Mikrocomputers unter einem etwas ungewöhnlichen Gesichtspunkt: Wie optimiere ich die Hardware bezüglich Ausnahmeverarbeitung? Das Problem ist mit unserem jetzigen Wissensstand nicht mehr sehr schwer zu lösen, da die CPU ja alle Randbedingungen vorgibt. Wir müssen sie nur im Design der Hardware gebührend würdigen.

Anforderungen an die Adreßdecodierung

Nach einem externen Reset, etwa beim Einschalten des Systems, liest der Prozessor als erstes unter den Adressen 0...7 den Startwert des Supervisor-Stackpointers und des Programmzählers. Es bietet sich daher an, auf den untersten Adressen den EPROM-Bereich anzuordnen, da der Programmzähler üblicherweise auch auf eine Initialisierungsroutine im EPROM zeigt. Der Startwert des Stackpointers muß natürlich auf einen RAM-Bereich zeigen. In einem modernen Computersystem wäre es schön, wenn das verfügbare RAM erst während der Initialisierungsphase gesucht und erst dann der Stackpointer geladen würde. Dummerweise aber müssen alle RAM-Suchroutinen die Busfehler-Ausnahme

benutzen, und dazu benötigt man bereits den Stack.

Womit wir ganz zwanglos bei der Tabelle der Ausnahmeverarbeitungsvektoren wären. Bei einem anständigen Computer gehört diese Tabelle ins RAM, unwiderruflich und ohne Ausflüchte! In 99 % aller Fälle werden einige Vektoren vom Betriebssystem oder vom Anwender während des Programmlaufs verändert; wenn dann die Ausnahmeverarbeitung erst über eine vordefinierte Tabelle im EPROM auf eine Hilfsroutine (meist auch im EPROM) verzweigt, die ihrerseits eine zweite Tabelle (diesmal im RAM) mit den derzeit gültigen Vektoren benötigt, um die eigentliche Ausnahmeverarbeitungsroutine zu finden, dann hat der Hardware-Designer ganz entschieden fünf Minuten zu wenig nachgedacht. Speziell wenn Interrupts im System herumgeistern, besteht der Witz ja gerade darin, die gesamte Unterbrechung so kurz wie möglich zu halten und die Reaktion auf dem schnellsten Weg auszulösen.

In 68010- und 68020-Systemen bereitet es keine Mühe, die Tabelle im RAM abzulegen, da die Prozessoren dieses hübsche Vektorbasisregister besitzen. Es können sogar mehrere Tabellen vorhanden sein, um blitzschnell den Kontext zu wechseln. Etwas aufwendiger gestaltet sich die Angelegenheit, wenn ein 68000 das Sagen hat. Im linearen Adreßbereich der CPU schließen die Ausnahmevektoren nahtlos am Resetvektor an – und der liegt ja nun schlauerweise im EPROM. Diese widersprüchliche Forderung ist wohl auch schon den Erfindern des 68000 aufgefallen, deshalb liegt der Resetvektor im Supervisor-Programmbereich und alle übrigen Ausnahmevektoren im Supervisor-Datenbereich. Durch Berücksichtigung der Funktionscode-Leitungen in der Adreß-Decodierung kann deshalb dieses Problem leicht aus der Welt geschafft werden. Eine andere, häufig angewandte Methode besteht darin, nach der Initialisierungsphase den EPROM-Bereich per Software auszublenden und durch einen RAM-Bereich zu ersetzen. Diese Lösung ist hardwaremäßig jedoch aufwendiger. Die letzte (und billigste) Möglichkeit, nämlich das eingangs erwähnte Führen einer zweiten Tabelle, ist keine Lösung, sondern ein Krampf.

Wer träumt nicht davon, seinen Rechner im vollen Adreßbereich mit Speicher auszustatten (abzüglich einer unbedeutenden Handvoll I/O-Adressen)? Abgesehen vom zur Zeit nicht unbeträchtlichen finanziellen Aufwand möge der geneigte Leser seinen Blick nochmals auf den ersten Teil dieses Beitrages richten: Die obersten 16 Adressen sind für die Interrupt-Bestätigung reserviert! Was glauben Sie, wird die CPU unternehmen, wenn während des Pseudo-Lesezyklus einer Interrupt-Bestätigung die Hardware ordnungsgemäß eine Vektornummer auf den Datenbus legt und gleichzeitig Ihre neuerworbene 4-MByte-RAM-Karte ihren Senf dazugibt, nur weil sie sich ebenfalls angesprochen fühlt? Die Frage ist leicht zu beantworten: Was auch immer Sie erwarten, die CPU wird sicher etwas anderes tun, nämlich Ihnen etwas husten. Und die Moral von der Geschichte: Man meide den obersten Adreßbereich, es sei denn – und hier kommt wieder die geniale Lösung – die Adreß-Decodierung berücksichtigt die Funktionscode-Leitungen (Tabelle 2, erster Teil), damit die Speicherkarte brav den Mund hält, wenn der CPU-Bereich dran ist.

Merke: Die Funktionscode-Leitungen sind nicht nur purer Luxus, sie können bei der Adreß-Decodierung an Stellen benötigt werden, an die man zunächst gar nicht gedacht hat.

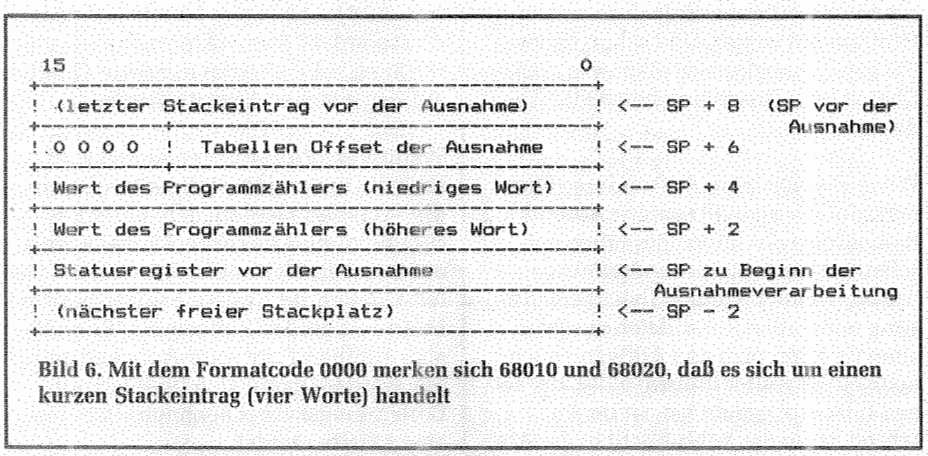


Bild 6. Mit dem Formatcode 0000 merken sich 68010 und 68020, daß es sich um einen kurzen Stackeintrag (vier Worte) handelt

Diese Überlegungen gelten für einfache Computersysteme; über die Komplikationen, die etwa durch den Einsatz einer MMU (Memory Management Unit) mit der Übersetzung von logischen in physikalische Adressen entstehen, wollen wir hier großzügig schweigen.

Reset- und Busfehler-Signal

Es gibt zwei Fälle, in denen ein externer Reset des Computers nötig wird: Beim Einschalten der Betriebsspannung und wenn ein entnervter Benutzer eine entsprechend bezeichnete Taste betätigt. Ein Busfehler tritt auf, wenn die CPU eine Adresse auf den Adreßbus gelegt hat und sich nach einer gewissen Zeitspanne noch niemand angesprochen fühlt. Sowohl beim Reset als auch bei Busfehlern muß ein Signal mit gewissen Eigenschaften erzeugt werden.
Dieser Aufgabe kann man sich sauber und elegant unter Verwendung von Zählern und Gattern entledigen (synchrone Methode) oder mit Hilfe von Monoflops (wer das tut, weiß genau, daß er sich eigentlich schämen müßte). Wie auch immer, diese beiden Signale sind lebenswichtig für den Mikrocomputer und dürfen in keinem System fehlen.

Interrupt-Erzeugung

Die meisten Überlegungen zum Hardware-Design müssen sicherlich in die Handhabung von Interrupts gesteckt werden. Ein Einplatinenrechner wird vielleicht nur einen oder zwei Interruptquellen haben; ein größeres System kann viele Quellen besitzen, die sich dann verschiedene Interrupt-Level teilen müssen. Je nach Intelligenz der Peripheriebausteine ist eine Erzeugung von externen Interruptvektoren einfach zu implementieren oder man muß mit Autovektoren auskommen.

Verhältnismäßig einfach liegen die Dinge, wenn weniger als sieben (bzw. drei beim 68008) unterschiedliche Interruptquellen vorhanden sind. Dann kann jedem Erzeuger eine Interruptstufe zugeteilt werden, ohne daß sich zwei gegenseitig stören. Level 7 (nicht maskierbar) sollte man nur verwenden, wenn man genau weiß, warum. Bei solch kleinen Systemen ist es normalerweise völlig ausreichend, mit Autovektoren zu arbeiten. Der Hardware-Designer kann sich das Leben einfach machen, denn er muß während der Interrupt-Bestätigung nur ein $\overline{\text{VPA}}$-Signal erzeugen. Wird ein 68010 oder 68020 eingesetzt, so müssen in dieser Decodierung neben den Funk-

tionscode- auch die Adreßleitungen ab A4 aufwärts enthalten sein, da diese Prozessoren mit dem Funktionscode 111 noch andere Tätigkeiten signalisieren. Die zeitliche Referenz dieser Signale wird durch den Adreß-Strobe ($\overline{AS}$) geliefert.

Diese einfache Erzeugung des $\overline{VPA}$-Signals hat noch einen großen Vorteil: Es können durchaus mehrere (hardwaremäßig unterschiedliche) Interrupts auf demselben Level auftreten. Durch Verwendung von Treibern mit offenen Kollektoren können problemlos mehrere Quellen auf derselben Interruptleitung zusammengeschaltet werden. Die Bestätigung des Autovektors erfolgt zentral, und es ist Aufgabe der Interrupt-Service-Routine, die in Frage kommenden Quellen abzufragen und zu bedienen. Im Grunde genommen handelt es sich hier um eine Kombination aus Interrupt und Polling. Der Nachteil besteht darin, daß das Ganze eben nur mit Autovektoren funktioniert.

In großen Computersystemen mit sehr vielen Interruptquellen kann das Abfragen aller Außenstellen, die auf dem gleichen Level liegen, zu unerwünscht langen Verzögerungen führen. In diesem Fall ist der Vektorinterrupt vorzuziehen, der von den meisten komplexen Peripheriebausteinen der Familie 680XX unterstützt wird. Die Zeiteinsparung in der Software muß jedoch mit einem erhöhten Aufwand in der Hardware ausgeglichen werden. Da nun jede Quelle ihre eigene Vektornummer abliefert, wird ohne zusätzlichen Verwaltungsaufwand gleich in die richtige Ausnahmeverarbeitung verzweigt. Der Zeitgewinn kann beträchtlich sein.

Vektor-Komplikationen

Alle denkbaren Kombinationen lassen sich auf den Fall zurückführen, daß zwei Vektorinterrupts gleichzeitig auftreten. Wenn sie verschiedene Prioritäten besitzen, ist das Problem schnell gelöst: Für jede Interruptquelle muß die Bestätigung durch die CPU separat und vollständig decodiert werden, das bedeutet die zusätzliche Auswertung der Adreßleitungen A1...A3, die den akzeptierten Level angeben. Dadurch wird sichergestellt, daß nur die Quelle mit der höchsten Priorität ihre Vektornummer auf den Datenbus legt.

Wenn zusätzlich noch Autovektorinterrupts auf anderen Leveln möglich sind, muß die Erzeugung des $\overline{VPA}$-Signals selbstverständlich auch individuell für jede Priorität, also unter Beachtung der Adreßleitungen A1...A3 erfolgen. Sonst wird während des Vektorlesezyklus die besagte Einfach-Decodierung vorschlagen, man könne doch genauso gut den Autorvektor nehmen.

```
 15                                                  0
+-------------------------------------------------+
! (letzter Stackeintrag vor der Ausnahme)        ! <-- SP + 12 (SP vor der
+-----------+-------------------------------------+          Ausnahme)
! Adresse der Instruktion, (niedriges Wort)      ! <-- SP + 10
+-die die Ausnahmebedingung---------------------+
! herbeigeführt hat          (höheres   Wort)    ! <-- SP + 8
+-----------+-------------------------------------+
! 0 0 1 0  !  Tabellen Offset der Ausnahme        ! <-- SP + 6
+-----------+-------------------------------------+
! Wert des Programmzählers (niedriges Wort)      ! <-- SP + 4
+-------------------------------------------------+
! Wert des Programmzählers (höheres Wort)        ! <-- SP + 2
+-------------------------------------------------+
! Statusregister vor der Ausnahme                ! <-- SP zu Beginn der
+-------------------------------------------------+     Ausnahmeverarbeitung
! (nächster freier Stackplatz)                   ! <-- SP - 2
+-------------------------------------------------+
```

Bild 7. Diese Stackeinteilung verwendet der 68020, wenn es sich um eine Ausnahme handelt, die per Befehl eingeleitet wurde

Das Ganze ist natürlich ein Schuß in den Ofen, wenn die zwei Vektorinterrupts auf derselben Prioritätsstufe liegen. Dann fühlen sich während der Bestätigung beide angesprochen, und die CPU erhält so etwas wie das logische Und der beiden Vektornummern, was dann weiter geschieht, muß wohl nicht näher erläutert werden. Die Lösung des Problems ist wahrhaftig nicht neu, stammt sie doch aus den Anfängen der Computerei, als eine anständige CPU sowieso nur einen Interrupt-Eingang besaß: Alle Quellen werden in einer Reihe hintereinander geschaltet (figürlich natürlich), wobei der wichtigste Kandidat vorne steht. Wenn die Interruptbestätigung der CPU kommt, schaut der Vorderste nach, ob er überhaupt ein Interrüptchen angefordert hatte. Wenn ja, dann legt er seinen Vektor auf den Datenbus und beendet den Pseudo-Lesezyklus des Prozessors. Wenn nicht, dann reicht er die schöne Bestätigung einfach an seinen Hintermann weiter. Und schon haben Sie ganz zwanglos begriffen, was „Daisy Chain" (= „Gänseblümchen-Kette") bedeutet.

Wenn Sie sich beim Aufbau Ihres Rechners einfach nicht für eine bestimmte Reihenfolge der Interruptquellen entscheiden können, oder anders ausgedrückt, wenn die totale Gleichberechtigung unter allen Kandidaten ausgebrochen ist, kann man den Aufwand auch noch etwas weiter treiben. Jeder, der seinen Interrupt gehabt hat, ist im Anschluß daran das Schlußlicht in der Abfrage-Kette. So etwas nennt man „Round Robin"-(= „Ringelreihn"-)Methode. Obwohl diese Begriffe aus der Computersteinzeit stammen, sind sie absolut modern. Beispielsweise schlagen sich Konstrukteure lokaler Netzwerke mit genau dem gleichen Problem herum: Wie vermeide ich Chaos auf meinem Bus.

Um den Hardware-Aufwand wirklich maximal zu gestalten, können jetzt sieben solcher Ketten aufgebaut werden, für jede Prioritätsstufe eine. Man darf natürlich auch Interruptquellen einordnen, die „nur" autovektorfähig sind, der CPU ist das ziemlich egal. Aber wie gesagt: So etwas geht auch einfacher, wenn nur Autointerrupts für die betreffenden Prioritäten vorgesehen sind.

Faustregeln

Bei wenigen Interruptquellen im System werden nur Autovektoren verwendet. Für die Decodierung des gemeinsamen $\overline{VPA}$-Signals ($\overline{AVEC}$ beim 68020) werden unbedingt die Funktionscode-Leitungen benötigt. Soll das System für 68010 und 68020 tauglich sein, so müssen zusätzlich die Adreßleitungen A4...A23 (...A31 beim 68020) mit decodiert werden. Autovektoren funktionieren auch bei mehreren Interrupt-Quellen auf der gleichen Prioritätsstufe.

Wenn Vektorinterrupts möglich sind, muß die Interruptbestätigung vollständig decodiert werden, d. h. unter Berücksichtigung des akzeptierten Levels auf den Adreßleitungen A1...A3. Bei vielen Quellen im System bringen Vektorinterrupts großen Zeitgewinn. Es muß aber erhöhter Hardware-Aufwand getrieben werden, damit sich im Interrupt-Bestätigungszyklus auch wirklich nur eine Quelle meldet.

Programmierers Lust und Frust

Aus der Sicht des Programmierers müssen für jede Ausnahmeverarbeitung zwei völlig getrennte Phasen unterschieden werden: Bevor die Ausnahme überhaupt auftreten darf, muß eine entsprechende Routine im System installiert sein, die dann die Reaktion auf die Ausnahme vornimmt. Vor allem muß ein Zeiger auf den Start dieses Programmstücks in die Vektortabelle eingetragen sein.

Die zweite, davon unabhängige Phase beginnt jedesmal dann, wenn die Ausnahmebedingung wirklich auftritt. Die CPU führt nun alle Schritte aus, die sich der Programmierer vorher so mühevoll überlegt hat, und mit etwas Glück entspricht das Resultat sogar seinen Erwartungen.

Initialisierung der Ausnahmeverarbeitung

Da die Einleitung wieder so kompliziert klingt, hier ein einfaches Beispiel: Jeder Computer benötigt ein Startprogramm, das er nach dem Einschalten der Stromversorgung durchläuft. Aus relativ einleuchtenden Gründen befindet sich dieses Programm in einem nicht flüchtigen Speicher (z. B. im EPROM). Die Anfangsadresse des Programmes ist gerade der Startwert des Programmzählers und der ist ja Bestandteil des Reset-Vektors in der Vektortabelle. Die Initialisierungsphase ist hier also während des Zusammenbaus des Computers erfolgt, nämlich indem das EPROM programmiert und in seinen Sockel gesteckt wurde.

In vielen Fällen besteht die erste und wichtigste Aufgabe des Startprogramms im Initialisieren der übrigen Ausnahmevektoren. Um nur einige zu nennen: die Vektoren für Busfehler und Spurious Interrupt werden mit den Adressen geeigneter (eventuell provisorischer) Prozeduren geladen, die sich häufig auch im EPROM befinden, denn noch bevor das Betriebssystem hochgefahren ist, können diese Ausnahmen hereinfunken. Später kann das Betriebssystem die Zeiger ja immer noch auf eine schönere und komfortablere Routine umlenken, denn die letztendlich wirksame Tabelle mit den Ausnahme-Vektoren liegt ja wie gesagt im RAM, auch beim 68000, unwiderruflich!

Dem Programmierer stellen sich nun zwei Probleme: Zum einen muß er herausfinden, ab welcher Adresse im Speicher der Loader oder Linker des Betriebssystems seine eben geschriebene Ausnahmeverarbeitungsroutine abgelegt hat. In vielen Fällen ist dazu eine intime Kenntnis des Betriebssystems notwendig, andererseits gibt es auch Hochsprachen, die dieses Begehren perfekt unterstützen. Nehmen wir zum Beispiel Modula-2: Es gibt dort den wunderschönen Typ „Prozedur" und einer Variablen dieses Typs kann man dann die Ausnahmeverarbeitungsroutine zuweisen. Wenn die Gedankengänge des Compilerbauers nicht allzu hinterlistig waren, besteht die interne Darstellung dieses Typs aus einer 32-Bit-Zahl – und zwar gerade aus der gesuchten Startadresse der Routine. Die zweite Schwierigkeit besteht im Finden der Vektortabelle. Bei 68010 und 68020 gibt es das Vektorbasisregister, dem man zumindest auf Assembler-Ebene zu Leibe rücken kann, und das Problem ist gelöst. Mit dem 68000 ist es etwas mühsamer. Wenn keine weiteren Informationen über den Aufbau des Betriebssystems vorliegen und wenn die ursprüngliche Vektortabelle doch im EPROM steht, kann man sich mit Hilfe eines Disassemblers aus der Affäre ziehen. Soll etwa der Interrupt-Autovektor Nr. 2 installiert werden, dann schaut man zuerst nach, welche Adresse in den Speicherplätzen 104...107 (Tabelle 1, 1. Teil) abgelegt ist. Von da aus gelangt man durch schrittweises Vorgehen und Disassemblieren irgendwann zu einem Programmstück, das die endgültige Referenz auf die RAM-Vektortabelle enthält. Während damit für die meisten Ausnahmen alle Vorbereitungen beendet sind muß ein Interrupt noch speziell freigegeben werden. In vielen Fällen, besonders bei intelligenten Peripheriebausteinen, sagt man zunächst der Peripherie Bescheid, daß sie jetzt darf, wenn sie will. Anschließend wird die Interrupt-Maske im Statusregister der CPU so gesetzt, daß der Interrupt die Chance bekommt, gehört zu werden. Das geschieht nun nicht etwa, indem man ganz brutal die gewünschte Priorität ins Register donnert (ein sehr beliebter Fehler), sondern man sieht tunlichst nach, welche Priorität eigentlich gerade zugelassen ist. Wenn nämlich bereits früher ein Interrupt niedrigerer Priorität freigegeben worden ist, würde dem durch die Holzhammer-Methode ziemlich herzlos das Wasser abgegraben, und so etwas tut man doch nicht, oder?

Im Fall eines einigermaßen intelligenten Peripheriebausteins (etwa aus der Familie 68xxx oder 63xxx) muß eventuell noch dessen Interruptvektorregister mit der gewünschten Vektornummer geladen werden. Für nicht initialisierte Vektorinterrupts ist in der Tabelle 1, 1. Teil, die Nummer 15 reserviert und alle Mitglieder der Familie 68xxx geben nach einem Reset diese Vektornummer ab, solange sie nicht eines Besseren belehrt wurden.

Können, Dürfen und vor allem Lassen

Tritt die Ausnahmebedingung nun wirklich einmal auf, ob gewünscht oder gefürchtet, so landet die CPU über kurz oder lang in der ordentlich installierten Routine. Was sie da tut, hängt natürlich von der Aufgabenstellung ab. Aber ein paar allgemeine Richtlinien sind sicher angebracht.

Als erstes muß der Inhalt aller CPU-Register, die in der Routine selbst benötigt werden, gerettet werden, und zwar auf den Stack. Sehr häufig findet man den Fehler, daß ein Registerinhalt in einer festen Speicheradresse abgelegt wird. Und da man ja ein bißchen haushalten muß, nutzt eine andere Ausnahmeverarbeitung diesen Speicherplatz gleich mit. Das geht dann vielleicht sogar eine ganze Zeit lang gut, nämlich solange beide Routinen auf der gleichen Priorität liegen und sich somit gegenseitig nicht unterbrechen können, aber wenn man dann nach Wochen die Priorität der einen Routine ändert, ergeben sich plötzlich ganz unerklärliche Fehler...

Natürlich müssen vor Beendigung der Ausnahmeroutine die Register wieder in ihren alten Zustand versetzt werden, wenn möglich ohne sie zu verwechseln. Durch Verwendung des MOVEM-Befehls kann fast nichts schiefgehen. Apropos Stack: Wenn jemand glaubt, er müsse das Adreßregister A7 für etwas anderes verwenden und könne den Inhalt mal schnell zwischenspeichern, dann ist er selber schuld. Sowie eine Ausnahmebedingung auftritt, fängt der Prozessor an, mit Hilfe des Supervisor-Stackpointers irgendwelche Daten zu retten und es ist ihm dabei ziemlich egal, wohin A7 in diesem Moment zeigt. Oft werden in Ausnahmeverarbeitungen gewisse Informationen vom Stack benötigt. Dazu hat man bequemen Zugriff mit Push und Pop, indirekter, indizierter, relativer oder sonst irgendeiner Adressierung, aber niemals sollte der Stackpointer direkt verändert werden. Notfalls kopiert man A7 in ein anderes Register, das man dann ungestraft manipu-

lieren darf. Mit dem Link-Befehl spart man dabei das vorherige Retten dieses Registers. Mit dem Anwender-Stapelzeiger kann man natürlich beliebig (oder jedenfalls beinahe) herumturnen, aber bei System- oder Interrupt-Stackpointer sind wir nur Untermieter, und die Hausordnung ist streng!

Merke: Die nächste Ausnahme kommt bestimmt und normalerweise gerade dann, wenn man sie am wenigsten erwartet. Wenn dann der Stackpointer nicht auf den Stack zeigt, dann geht die Angelegenheit mit ziemlicher Sicherheit in die H§$&.

Sofern die Hardware nicht schon dafür sorgt, muß eine Interrupt-Service-Routine veranlassen, daß die Quelle ihre Anforderung auch zurücknimmt, im anderen Falle befindet man sich nach dem Ende der Ausnahmeverarbeitung gleich wieder an deren Anfang und das geht so weiter, bis jemand die Geduld verliert. Bei einem Bus- oder Adreßfehler und einem Prozessor 68000 muß man den Stack selbst in Ordnung bringen.
Noch ein Wort zum Schluß: Niemand verlangt, daß eine Ausnahmeverarbeitungsroutine komplett in Assembler geschrieben wird. Beim Initialisieren, Retten der Registerinhalte, Reinstallieren sowie beim RTE-Befehl wird es sich nicht vermeiden lassen, aber sonst spricht (außer vielleicht Zeitgründen) nichts gegen die Verwendung von Hochsprachen. Vorausgesetzt, man weiß, wohin die Prozedur oder Funktion vom Betriebssystem geladen wurde.

Polling von Interrupt-Quellen

Sehr häufig tritt der Fall ein, daß für einen Interrupt mehrere verschiedene Quellen in Betracht kommen, sei es, daß ein komplexer Peripheriebaustein mehr als eine mögliche Ausnahmebedingung herbeiführen kann oder daß sich verschiedene Hardware-Quellen die gleiche Priorität teilen. Damit die Ausnahmeverarbeitungsroutine den eigentlichen Verursacher des Interrupts herausfinden kann, werden nacheinander die Statusregister der einzelnen Kandidaten abgefragt. Auch dabei schleicht sich schnell ein bestimmter Fehler ein: Wenn ein spezielles Statusbit (etwa für eine Handshake-Leitung in 6522 oder 68230) gesetzt ist, bedeutet das noch nicht, daß der Interrupt von diesem Baustein kam; ausschlaggebend ist in diesem Fall, ob dieser Interrupt überhaupt durch die Baustein-interne Maske freigegeben war, und das muß der Programmierer nun leider selbst überprüfen. Wer hier nicht aufpaßt, läßt Interrupts bearbeiten, die eigentlich gar nicht durchkommen sollten.

Ein anderer beliebter Fehler ist es, beim Bearbeiten eines Interrupts aus Versehen gleich eine zweite Anforderung im gleichen Baustein zu löschen. Also: In jeder Service-Routine nur die Quelle zurücksetzen, die auch wirklich schon bedient wurde. Der gleiche Grundsatz gilt natürlich auch für die Freigabe in der Interrupt-Maske des Bausteins; es ist gefährlich, immer pauschal alle möglichen Quellen freizugeben oder zu sperren, irgend ein Programmstück hat ja vielleicht gute Gründe, einen Interrupt zeitweilig abzuklemmen oder zu erlauben.
Durch die Reihenfolge, in der die möglichen Quellen abgefragt werden, ist es auf einfache Art möglich, Software-Prioritäten zu implementieren: Die wichtigsten Interrupts werden immer zuerst geprüft.

Spruch des Monats

„Es sieht so aus als mache die Wirklichkeit einen Umweg über die Träumer."

Titel eines Buches von Jörg Löhken, das im Verlag „Unter uns" erschienen ist

Unterschiede

Jetzt sind wir auch soweit, die eingangs gestellte Frage nach den Unterschieden zwischen den einzelnen Familienmitgliedern bezüglich Ausnahmeverarbeitung beantworten zu können. Von allen Prozessoren wird im wesentlichen die gleiche Hardware benötigt (von den verschiedenen Busbreiten einmal abgesehen) und wenn von Anfang an die Interrupt-Bestätigung vollständig decodiert wird (Funktionscode- und alle Adreßleitungen), kann gefahrlos ein Prozessor gegen den anderen ausgetauscht werden. Nicht ganz so einfach liegen die Verhältnisse bei der Software. 68010 und 68020 sind sehr gut miteinander verträglich, aber es bestehen ziemlich starke Abweichungen zum 68000. Unterschiede und Gemeinsamkeiten lassen sich wie folgt zusammenfassen:

(1) Das fehlende Vektorbasisregister des 68000 erfordert eine andere Initialisierung der Ausnahmen.

(2) Das Stackformat des 68000 bei Bus- und Adreßfehlern ist grundsätzlich anders aufgebaut. Zur Fehlerbehandlung werden aber gerade die Informationen über die Ursache vom Stack gebraucht. Auch 68010 und 68020 besitzen unterschiedliche Stackformate und retten andere interne Zustände. Diese Ausnahmeverarbeitungen müssen also für jede CPU individuell programmiert werden.

(3) Bei allen anderen Ausnahmen werden Statusregister und Programmzähler an den gleichen Stellen auf dem Stack abgelegt (relativ zum Stackpointer). Wenn keine weiteren Informationen benötigt werden, sind dieselben Verarbeitungsroutinen auf allen Prozessoren lauffähig.

Zusammenfassend: Wie zu erwarten war, sind die Mikroprozessoren der Familie 680xx untereinander und übereinander kompatibel, aber...

Literatur:

[1] Datenblatt MC68008, Motorola 1982
[2] Datenblatt MC68000, Motorola 1985
[3] Datenblatt MC68010, Motorola 1983
[4] MC68020 Users Manual, Second Edition, Motorola 1985
[5] *Hilf, Werner/Nausch, Anton:* MC68000-Familie, Teil 1 und 2, Te-Wi-Verlag München 1984
[6] *Kane, Gerry:* 68000 Mikroprozessorhandbuch, McGraw-Hill Book Company GmbH, Hamburg 1981
[7] *Grohmann, Bernd/Eichler, Lutz:* Das Prozessorbuch zum 68000, Data Becker, Düsseldorf 1985
[8] *Johnson, Thomas L.:* A Comparison Of MC68000 Family Processors, Byte 9/1986, Seite 205

Oliver Häußler, Dieter Strauß

PC-Erweiterungskarten

Eine aktuelle Marktübersicht

Das Angebot an PC-Erweiterungskarten ist stark gewachsen, sowohl in der Zahl der Anbieter als auch der zusätzlichen Funktionen, die auf einer Karte integriert sind. Es gibt fast keine Anwendung, die man nicht mit einer Erweiterungskarte durchführen könnte. mc zeigt in dieser Marktübersicht eine breite Palette derzeit verfügbarer Erweiterungskarten.

Ursprünglich spielten die PC-Erweiterungskarten nur eine Lückenbüßerrolle, da sie Funktionen auf einer Karte integrierten, die zu einem Computer gehören sollten, jedoch im Original-PC nicht vorhanden waren. So fehlte z. B. eine Uhr mit Kalender und der Speicher war zu klein. Dieser Mangel wurde durch eine offene Systemarchitektur, die es erlaubt, auf den Erweiterungssteckplätzen Karten mit zusätzlichen Funktionen einzubauen, mehr als wettgemacht.
Der Erfolg des IBM-PCs und seiner vielen Nachbauten dürfte nicht in seiner Technik liegen – die bereits zum Zeitpunkt seines Erscheinens nicht die modernste war – sondern in seiner offenen Systemarchitektur und dem Wunsch der Anwender nach standardisierter Hard- und Software.

Was heißt EMS-kompatibel?

Im Zusammenhang mit den PC-Erweiterungskarten werden sehr viele Abkürzungen verwendet. mc erläutert ihre Bedeutung:

BAS: Bild-Austast-Synchronisations-Signal (composite video).

CGA: Color Graphics Adapter. Farbgrafik-Karte mit einer Auflösung von 320 × 200 Bildpunkten mit vier Farben. Anforderungen an den Monitor:
V-Sync: 60 Hz, H-Sync: 15,7 kHz

EGA: Enhanced Graphics Adapter. Farbgrafik-Karte mit einer Auflösung von 640 × 350 Bildpunkten mit bis zu 16 Farben. Anforderungen an den Monitor:
V-Sync: 60 Hz, H-Sync: 21,8 kHz (im 350 Grafikzeilen-Modus)

EEMS: Enhanced Expanded Memory Specification. Von AST Research verbesserter EMS. Im Unterschied zu EMS kann EEMS die logischen Speicherseiten im ganzen vom Mikroprozessor adressierbaren Speicherbereich unterbringen – also auch in den von MS-DOS kontrollierten Speicherbereich bis A000:0000H (640 KByte).

EMS: Expanded Memory Specification. Von den Firmen Lotus, Intel, Microsoft geschaffenes System der Speicherbank-Verwaltung. Bis zu 8 MByte können von Anwenderprogrammen angesprochen werden (unter MS-DOS). Der erweiterte Speicher dient den Anwenderprogrammen als Auslagerungsbereich für derzeit nicht benötigte Datenbereiche (keine Programme!). Der vom DOS nicht genutzte Speicherbereich von C000:0000H...D000:0000H wird in vier Speicherseiten mit je 16 KByte unterteilt. Diesem Speicherbereich, Page Frame genannt, werden vier logische Speicherseiten der EMS-Karte zugeordnet. Der Inhalt dieser vier Seiten der EMS-Karte wird in den Page Frame kopiert. Die Verwaltung des Page Frame übernimmt ein installierbarer Treiber, der Expanded Memory Manager (EMM). Speicherkarten nach EMS werden fast immer zusammen mit dem Treiber geliefert. Insgesamt kann EMS 512 Seiten verwalten.

Enhanced RGB: Karten mit diesem Ausgang liefern zusätzlich zu den unter RGB beschriebenen Signalen zwei zusätzliche Farbsignale – R' und B'. Damit sind 64 Farbkombinationen möglich. EGA-Karten verwenden Enhanced RGB.

Extended Memory: Der Speicherbereich oberhalb von einem Megabyte, auf den der 80286 im Protected-Mode zugreifen kann. Die derzeitig erhältlichen DOS-Versionen können diesen Speicherbereich nicht nutzen. Mit dem Betriebssystem Xenix ist dieser Bereich direkt auszuschöpfen.

FBAS: BAS-Signal mit zusätzlicher Farbinformation.

FFT: Fast Fourier Transformation. Schnelle Fouriertransformation. Wird zur Analyse von Signalen (Spektrumanalysator) gebraucht.

Hercules: Monochrome Grafik-Karte. Auflösung 720 × 348.
Anforderungen an den Monitor:
V-Sync: 50,9 Hz, H-Sync: 18,8 kHz

IEEE-488: Bussystem, das besonders in der Meßtechnik zum Verbinden intelligenter Meßsysteme verwendet wird.

LIM: Lotus-Intel-Microsoft. Diese drei Firmen haben den EMS-Standard geschaffen, der Anwenderprogrammen unter MS-DOS erlaubt, auf Speicherbereiche jenseits der 640-KByte-Grenze zuzugreifen.

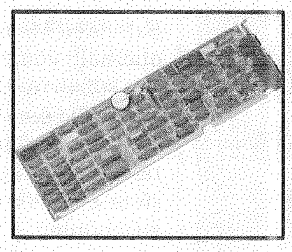
MDA: Monochrome Display Adapter. Video-Karte für den Textmodus.
Anforderungen an den Monitor:
V-Sync: 50 Hz, H-Sync: 18,2 kHz

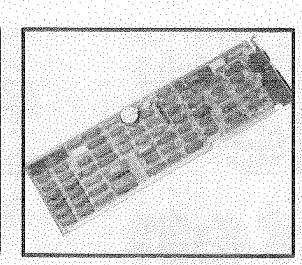
RGB: Karten mit RGB-Ausgang (digital) liefern die Farb-Signale (rot, grün, blau), ein Helligkeits-Signal und die Synchronisations-Signale V-Sync und H-Sync. Aus der Kombination der drei Farbsignale mit dem Helligkeits-Signal ergeben sich maximal 16 Farbkombinationen. Die Signale haben TTL-Pegel.

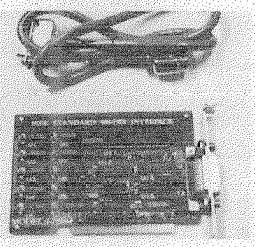
Preise und Anbieter

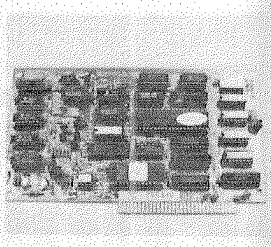
Die angegebenen Preise sind einschließlich Mehrwertsteuer und beziehen sich auf Mitteilungen der Hersteller. Viele Karten werden komplett mit Software geliefert. Dies ist in der Rubrik Software vermerkt.
Fast jede Karte integriert mehrere Funktionen. Es ist uns nicht gelungen, die Karten nach Funktionen sinnvoll zu ordnen. Fast jede Karte wäre bei diesem Ordnungsschema eine Multifunktionskarte. Wir haben daher die Karten alphabetisch nach den Namen der Anbieter geordnet. Viele Karten werden neben dem genannten Anbieter auch beim örtlichen Fachhandel erhältlich sein.

mc-marktübersicht

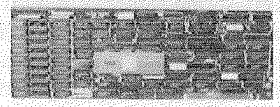


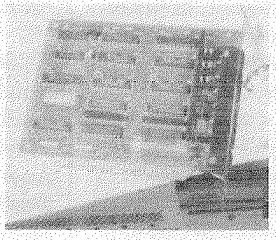


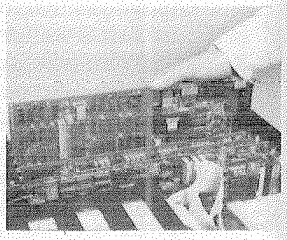




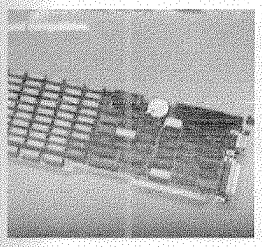
Typ	CT 6040	CT 6280	IEE 488	1,2 MB/360k
Hersteller bzw. Vertrieb	Add In Computer, München	Add In Computer, München	AD Computertechnik, Augsburg	AD Computertechnik, Augsburg
Preis (DM)	468	168	488	199
Funktion	Grafik	Speichererweiterung	Meßgerätesteuerung	Floppydisk-Controller
Einsetzbar in	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT/-AT und Kompatible	IBM-XT/-AT und Kompatible	IBM-XT/-AT und Kompatible
Speicher	–	max. 512 KByte	–	–
Massenspeicher	–	–	–	–
Grafik	Hercules-Standard	–	–	–
A/D, D/A	–	–	–	–
Uhr/Kalender	–	–	–	–
Schnittstellen	18 kHz Videoport, Centronics	–	IEEE 488	ST 506
Software	–	–	–	–
Bemerkungen	Wird von Lotus, Autocad, Framework, Symphony unterstützt	Adreßbereiche mit DIL-Schalter einstellbar	3 DMA-Kanäle, Talker-, Listener-, Controller-Funktion, kaskadierbar	Diskettenlaufwerke mit 360 KByte, 720 KByte, 1,2 MByte anschließbar



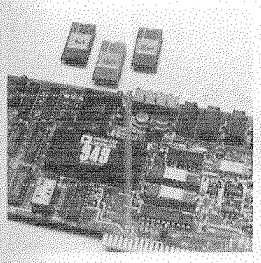




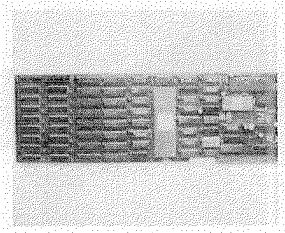
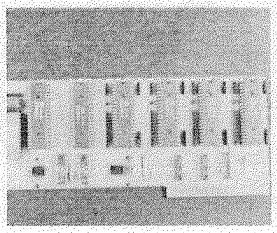
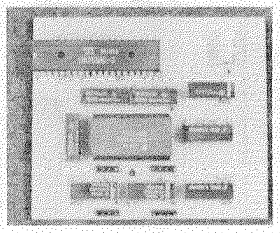
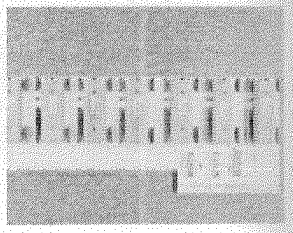
Typ	AM 616 Videotrax	AM-170	RTI 817	RTI 800
Hersteller bzw. Vertrieb	AM-Distribution, München	AM Distribution, München	Analog Devices, München	Analog Devices, München
Preis (DM)	1790	7410	1100	3210
Funktion	Datensicherung	Erweiterungskarte	Digital-E/A	A/D und Digital-E/A
Einsetzbar in	IBM-XT-/-AT und Kompatible	IBM-XT/-AT und Kompatible	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT/-AT und Kompatible
Speicher	4-KByte-Puffer	1 MByte	Übernahmespeicher	–
Massenspeicher	–	–	–	–
Grafik	–	–	–	–
A/D, D/A	–	–	–	12-Bit-A/D-Wandler
Uhr/Kalender	–	●	–	–
Schnittstellen	Videoausgang 75 Ω	–	–	–
Software	Bedienungsprogramm	Betriebssystem: AMOS	–	–
Bemerkungen	7 Optionen, für alle VHS, Beta, Video 2000	Bis zu 6 Anwender, Multitasking-Betriebssystem	24 Kanäle, Ausgänge rücklesbar, kompatibel zu Relaisbausteinen. Interrupt (IRQ2..IRQ7) auswählbar	8 differentielle oder 16 unipolare Analog-Eingabekanäle, Meßraten über 71 kHz, 3 Zähler/Zeitgeberkanäle, 8 Digital-Eingänge, 8 Digital-Ausgänge



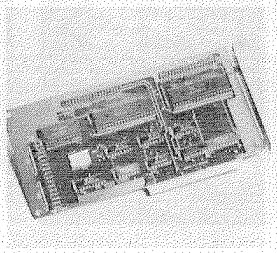
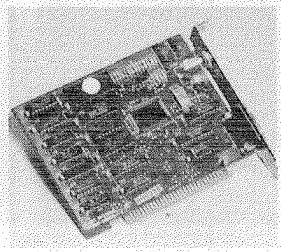
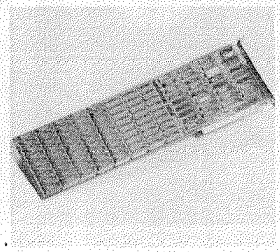
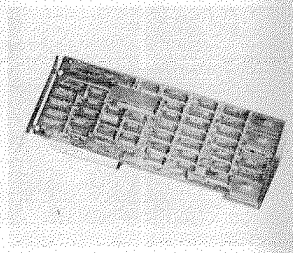
Multi-I/O-Aboveboard	Seriell-Parallel 3	Seriell-Parallel 4	Color-Grafik	Multi I/O
AD Computertechnik, Augsburg	AD Computertechnik, Augsburg	AD Computertechnik, Augsburg	AD Computertechnik, Augsburg	AD Computertechnik, Augsburg
630	185	235	155	235
E/A-Aboveboard	Schnittstellen	Schnittstellen	Grafikkarte	Multifunktion
Printspooler, Ramdisk	IBM-XT/-AT	IBM-PC/-AT	IBM-PC/-XT	IBM-PC/-XT
bis 2 MByte	–	–	–	–
–	–	–	–	–
–	–	–	640 × 200 oder 320 × 200	–
–	–	–	–	–
–	–	–	–	●
par. und ser. Port, Game Port	Centr., 1 ser.	2 × RS232, Centr.	RGB, BAS	ser., par.
Abovetreiber	–	–	–	–
Entspricht der LIM-EMS-Spezifikation	2. serieller Anschluß optional	Multilayer-board	kompatibel zum Industrie-standard	Für XT mit Floppy-controller, 2 ser. Port optional



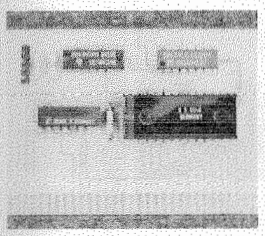
RTI 815	SixPak Premium/EGA	Advantage Premium	I/O-Mini-II	ATI Graphics Solution
Analog Devices, München	AST Research, Düsseldorf	AST Research, Düsseldorf	AST Research, Düsseldorf	Kolter, Erftsstadt
4100	1560	2070	433	850
A/D und Digital-E/A	Multifunktion	Multifunktion	Multifunktion	Grafik
IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT und Kompatible	IBM-AT und Kompatible	IBM-PC/-XT und Kompatible	IBM-PC/-XT/-AT und Kompatible
–	Bis 2 MByte erweiterbar	Bis 2 MByte erweiterbar	–	–
–	–	–	–	–
–	EGA, CGA, MDA od. Hercules	–	–	Hercules-, CGA-Standard
12-Bit-A/D-Wandler	–	–	–	–
–	●	●	●	–
	1 ser., 1 par.	1 par., 1 ser., Game Port	1 par., 1 ser.	1 ser./par. (optional)
	AST SuperPak-Software	AST SuperPak-Software	–	–
8 differentielle oder 16 unipolare Analog-Eingabekanäle, 8 Digital-Eingänge, 8 Digital-Ausgänge, zus. 2 Analog-Ausgänge	LIM-Standard	LIM-Standard	Kurze Steckkarte	Bildschirmdarstellung in 16 Graustufen, Farbgrafik Emulation

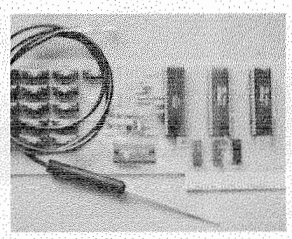
Typ	RDDK	Multiboard 86	A/D-Wandler	I/O-Platine 192 Bit
Hersteller bzw. Vertrieb	Bartscher, Eschwege	Bockstaller, Wehr	Bockstaller, Wehr	Bockstaller, Wehr
Preis (DM)	4560	1475	560	540
Funktion	Videodigitalisierer	A/D-, D/A-Wandler	Analog/Digital	Digital-E/A
Einsetzbar in	IBM-PC/-XT/-AT und Kompatible, PPC	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT und Kompatible	IBM-PC/-XT und Kompatible
Speicher	512-KByte-Bildspeicher	–	–	–
Massenspeicher	–	–	–	–
Grafik	256 Farben, 640 × 400, 1/25 Sek. Bilderfassung	–	–	–
A/D, D/A	8-Bit-D/A-Wandler	12-Bit-A/D mit Multiplexer, 12-Bit-D/A	12-Bit-A/D mit 16 Kanälen	–
Uhr/Kalender	–	–	–	–
Schnittstellen	Kamera, Videorecorder	–	optional	–
Software	Turbo-Pascal	Demosoftware und Treiber	–	–
Bemerkungen	Zoom in X- und Y-Richtung in 15 Stufen programmierbar, für Bilddokumentation u. -auswertung	Meßwerterfassung: bis zu 32 analoge Eingangsspannungen	760 DM mit Gehäuse	Für Steuerungen

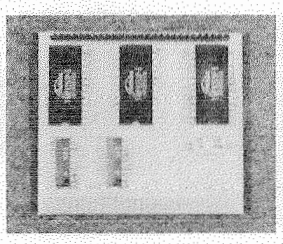
Typ	8255-VIA-Karte-3	Color Grafik TTL-4	Aboveboard 2 MByte – 5	Speedkarte-6
Hersteller bzw. Vertrieb	bsb Datentechnik, Köln	bsb Datentechnik, Köln	bsb Datentechnik, Köln	bsb Datentechnik, Köln
Preis (DM)	168	349	398 (ohne RAM)	479
Funktion	E/A-Karte	Grafik	Speichererweiterung	Beschleunigerkarte
Einsetzbar in	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT/-AT und Kompatible	IBM-XT/-AT und Kompatible	IBM-PC/-XT und Kompatible
Speicher	–	–	–	8-KByte-/16-KByte-Cache
Massenspeicher	–	–	Bis zu 2 MByte erweiterbar	–
Grafik	–	640×200 s/w, 320×200 4 Farben	–	–
A/D, D/A	–	–	–	–
Uhr/Kalender	–	–	–	–
Schnittstellen	–	RGB, Video-BAS, TTL monochrom	–	–
Software	–	–	RAM Disk, Druckerspooler	–
Bemerkungen	1 Zähler/Zeitgeber, E/A-Adressen einstellbar, 48 programmierbare Ein-/Ausgabeleitungen über 2×3 8-Bit-Kanäle	Textmodus 80×25 in 16 Farben	Preis ohne RAM, LIM-Standard	8086 (10 MHz), Leistungssteigerung 170–240 %



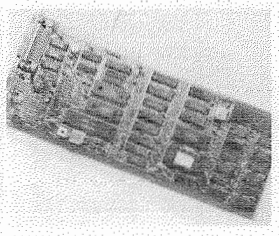
16-Bit-Up/Down-Counter
Bockstaller, Wehr
350
Vorwärts-/Rückwärtszähler
IBM-PC/-XT/-AT und Kompatible
–
–
–
–
–
RS 232
–
Pulsbreiten- und Frequenzverdoppelung möglich



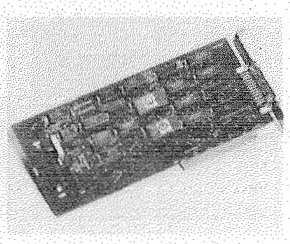
Thermoboard 86
Bockstaller, Wehr
980
Temperaturmessung
IBM-PC/-XT/-AT und Kompatible
–
–
–
–
–
–
Basic
Erfassung von 16 untersch. Temperaturen (−50 °C bis 150 °C), inkl. Meßfühler



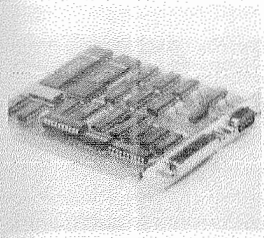
Printi
Bockstaller, Wehr
350
Zeitgeber/Zähler
IBM-PC/-XT/-AT und Kompatible
–
–
–
–
–
–
–
9 Zeitgeber/Zähler mit je 16 Bit, 4 Dekaden im BCD-Modus



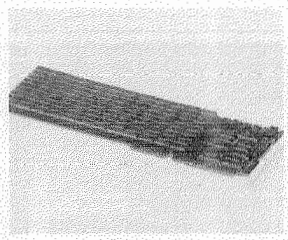
Btx-RAFI-1
bsb Datentechnik, Köln
1139
Btx-Karte
IBM-XT/-AT und Kompatible
–
–
RGB-Monitor oder SCART-TV
–
–
RGB und SCART
inkl.
DBP-03-Modem erforderlich, umschalten auf PRESTEL möglich



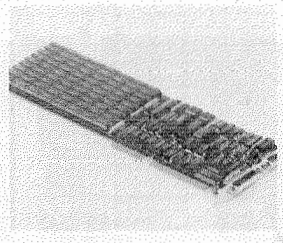
A/D-Wandler-2
bsb Datentechnik, Köln
269
A/D
IBM-PC/-XT/-AT und Kompatible
–
–
–
Unipolarer 12-Bit-A/D-Wandler mit 16 Kanälen
–
–
Basic-Demoprogramm
1 D/A-Kanal, E/A-Adressen einstellbar, Abtastrate >40 kHz



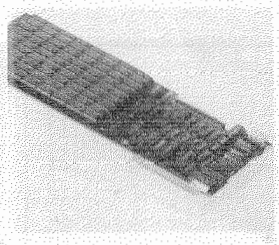
Multi-Port
Computer Elektronik Infosys, Bodenheim
495
Schnittstellen-Karte
IBM-PC/-XT/-AT und Kompatible
–
–
–
–
–
2 ser. V24, 1 par. Centronics
Druckerspooler, RAM-Disk
Centronics mit 2 alternativen Portadressen, jede Schnittstelle einzeln ausblendbar



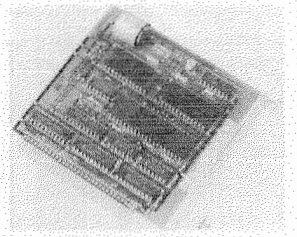
Mem AT+
Computer Elektronik Infosys, Bodenheim
1380
Speichererweiterung
IBM-AT und Kompatible
Bis 3 MByte erweiterbar
–
–
–
–
–
Betrieb ohne Wartezyklen möglich, mit 128 KByte RAM 1520 DM, mit 384 KByte RAM 1980 DM



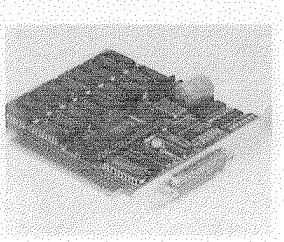
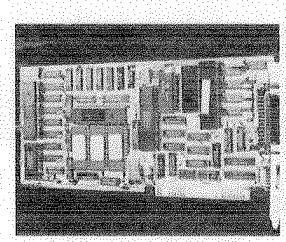
Multi 3 AT
Computer Elektronik Infosys, Bodenheim
1470
Multifunktionskarte
IBM-AT und Kompatible
128...384 KByte Basisbereich
0...2 MByte extended
–
–
–
2 ser. V.24, 1 par. Centr.
–
Betrieb ohne Wartezyklen möglich, mit 128 KByte RAM 1650 DM, mit 384 KByte RAM 2070 DM



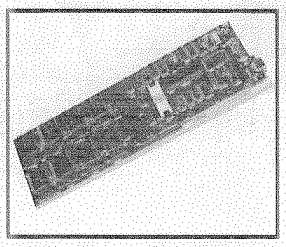
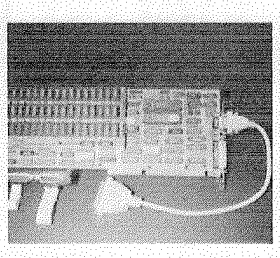
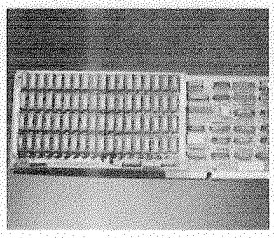
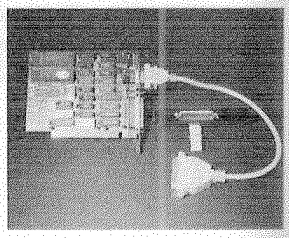
Mega RAM
Computer Elektronik Infosys, Bodenheim
995
Speichererweiterung
IBM-PC/-XT und Kompatible
Bis zu 8 MByte aufrüstbar
–
–
–
–
(optional)
–
–
Basisbereich bis 448 KByte aufrüstbar, mit 1 MByte RAM 1545 DM, mit 2 MByte RAM 2195 DM



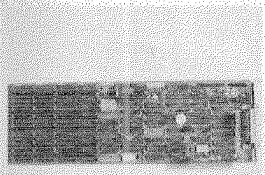
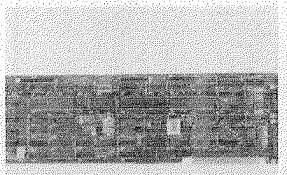
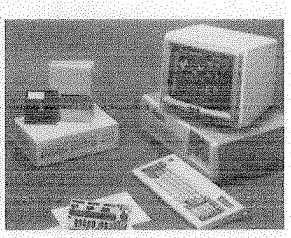
Elkey Nr. 1
Computer Elektronik Infosys, Bodenheim
rd. 600
Sicherheitskarte
IBM-PC/-XT/-AT und Kompatible
–
–
–
–
nicht beeinflußbar
–
Schutzmechanismen
5stelliger alphanum. Code, 10stellige Personalnummer, netzwerkfähig, kurze Steckkarte

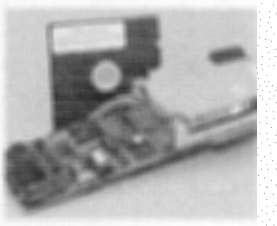



Typ	MultiRAM+Clock+Ser. Port	SDX X.25	PCC-CG-80	PCC-MG-GP
Hersteller bzw. Vertrieb	Computer Elektronik Infosys, Bodenheim	Comtes, Bremen	Cologne Tronix, Köln	Cologne Tronix, Köln
Preis (DM)	735	4440	117	163
Funktion	Multifunktionskarte	SNA-Adapter	Grafik	Grafik
Einsetzbar in	IBM-PC/-XT und Kompatible, PPC	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT und Kompatible	IBM-PC/-XT/-AT und Kompatible
Speicher	0...512 KByte	64 KByte	16-KByte-Bildspeicher	64-KByte-Bildspeicher
Massenspeicher	–	–	–	–
Grafik	–	–	640 × 200 s/w, 320 × 200 Farbe, 40/80 × 25	Hercules-kompatibel
A/D, D/A	–	–	–	–
Uhr/Kalender	●	–	–	–
Schnittstellen	ser. V.24	V.24 (X.21 bis), Centronics (optional)	RGBI, BAS	TTL-Ausgang, V-Sync und H-Sync
Software	Drucker, Treiber	SNA-3270-Emulation	–	–
Bemerkungen	Nicht benötigter Speicher ist ausblendbar, mit 64 KByte 655 DM, mit 128 KByte 725 DM, mit 512 KByte 1185 DM	DATEX-P, 1 Arbeitskanal, 1 Diagnosekanal, FTZ-Zulassung	Mit Motorola IC 6845	720 × 348 hochaufl. s/w-Grafik

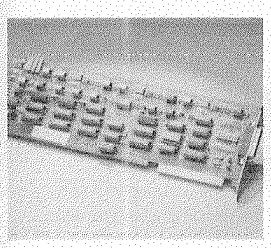
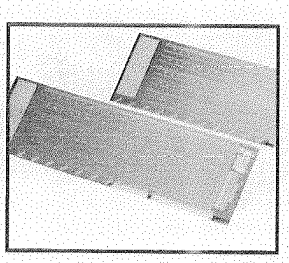
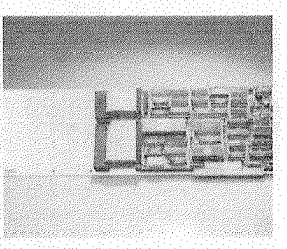
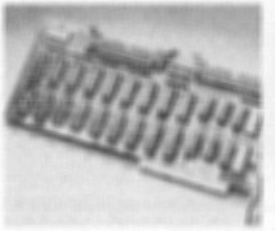
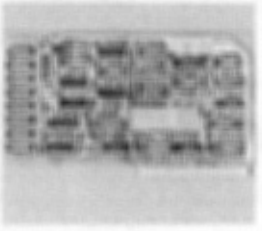





Typ	VHGR-F	MF-3000	Memo 4000	DIO 500 AT
Hersteller bzw. Vertrieb	Gorny, Schweskau	Hintz, Ottobrunn	Hintz, Ottobrunn	Hintz, Ottobrunn
Preis (DM)	1490	1440	1550	300
Funktion	Grafik	Multifunktion	Speichererweiterung	E/A-Karte
Einsetzbar in	IBM-PC/-XT und Kompatible	IBM-AT und Kompatible	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT/-AT und Kompatible
Speicher	–	–	–	–
Massenspeicher	–	Bis zu 1,5 MByte	2 MByte	–
Grafik	1024 × 1024, 8 Farben	–	–	–
A/D, D/A	–	–	–	–
Uhr/Kalender	–	–	–	–
Schnittstellen	RGBI, BAS	2 V.24, 2 par., Game Port	–	2 V.24, 1 par.
Software	optional	Optional: Above-Disk	–	–
Bemerkungen	Mit NEC-7220-Grafik-Controller	Speichererweiterung bis zu 3 MByte mit Aufsteckkarte möglich, belegt mit Aufsteckkarte 1½ Steckplätze	4 MByte Speicheraufrüstung mit zusätzlicher Aufsteckkarte möglich, 1½ Steckplätze	Kurze Erweiterungskarte

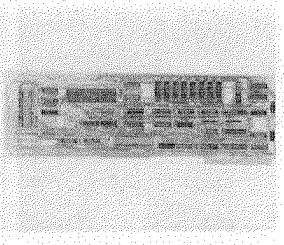




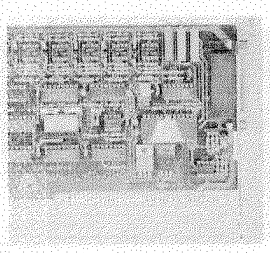


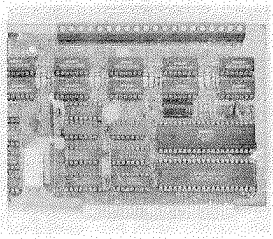
PCC-KPSU-384/0	PCC-KPSU-FDC	UMicro CIF	Oculus 100	Easy Card
Cologne Tronix, Köln	Cologne Tronix, Köln	Datalog, Mönchengladbach	Datalog, Mönchengladbach	DSM, München
196	177	2844	7866	1498
Multifunktion	Multifunktion	Meßwerterfassung	Bildverarbeitung	Festplattenkarte
IBM-PC/-XT und Kompatible	IBM-PC/-XT und Kompatible	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT/-AT und Kompatible
bis 384 KByte bestückbar	–	–	32-KByte-Bildspeicher	–
–	–	–	–	20 MByte
–	–	–	512 × 512 Punkte s/w, bis zu 256 Graustufen	–
–	–	12-Bit-A/D-Wandler	Video-A/D-Wandler	–
●	●	–	–	–
par., ser., Game Port	par., ser., Floppy, Game Port	8/48 Digital-Ein-/Ausgänge, 8/16 Analog-Eingänge	4 Kameras (CCIR oder RS-170-Norm)	–
–	–	Demo in Basic/Turbo-Pascal	Treiber	inkl. Dienstprogramm
–	Mit Floppy-Controller	Zur Meßwerterfassung und Steuerung, 4 programmierbare Meßbereiche, 10 000 Messungen/s	Für Standardkameras, Bildverarbeitung, 50 Bilder/s, 128 Graustufen, 256 Farben optional	Erweiterungskarte mit 3½"-Festplatte

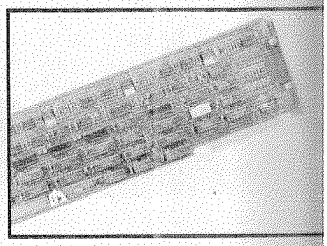






IM-ADDA	PTP-Card	IM-HDQ	IM-TTL 64	D 401
Impec, Tübingen	Impec, Tübingen	Impec, Tübingen	Impec, Tübingen	iSystem, Dachau
1220	114	741	1044	775
A/D-, D/A-Wandler	Prototypen-Karte	Festplatten-Controller	E/A-Karte	Z80-CP/M-Coprozessor
IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT (PTA-Card für AT) und Kompatible	–	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT/-AT und Kompatible
–	–	–	–	64 KByte
–	–	–	–	–
–	–	–	–	–
12-Bit-A/D-Wandler mit 8 Kanälen, 12 Bit D/A	–	–	–	–
–	–	–	–	–
–	–	ST-506	–	–
6sprach. Demoprogr.	–	–	inkl. Testprogramm	CP/M-Emulator
Analogausgangssignal 0...10 V, Prozeßstromausgang 0...40 mA	Karte zur Fertigung von Prototypen oder Kleinserien	Controller für zwei 3½"-Festplatten mit je 32 MByte	TTL-E/A-Karte mit 8 E/A-Kanälen, 64 E/A-Leitungen	Bildschirmemulation für Kaypro, inkl. Dienstprogramme

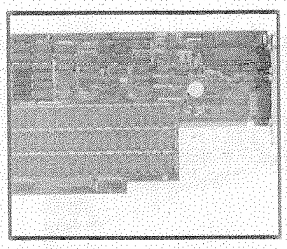


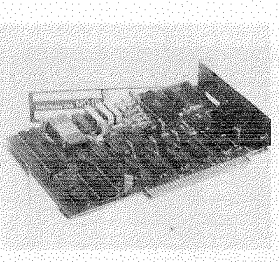


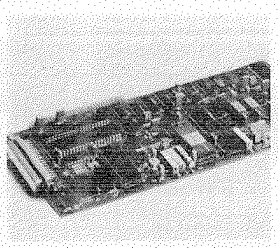


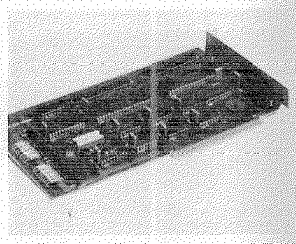


Typ	D 64180A	DM-P011	DM-P014	DM-P005
Hersteller bzw. Vertrieb	iSystem, Dachau	Köpke, Asbach	Köpke, Asbach	Köpke
Preis (DM)	ab 1653	1600	1560	1480
Funktion	CP/M-Coprozessor	A/D-Wandler	Temperaturmessung	A/D-D/A-Wandler
Einsetzbar in	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT und Kompatible	IBM-PC/-XT und Kompatible	IBM-PC-XT
Speicher	256 KByte, auf 512 KByte erweiterbar	–	–	–
Massenspeicher	–	–	–	–
Grafik	–	–	–	–
A/D, D/A	–	8-Kanal-12-Bit-A/D-Wandler	8-Kanal-12-Bit-A/D-Wandler	8-Kanal-12-Bit-AD-, 12-Bit-D/A-Wandler
Uhr/Kalender	●	–	–	–
Schnittstellen	2 V.24	–	–	–
Software	Software-Entwicklungskit	–	–	–
Bemerkungen	2 Zähler/Zeitgeber, 2 DMA-Kanäle	Wandlungszeit: 35 µs	30 Messungen/s, Dual-Slope-Meßverfahren	A/D; D/A-Wandlerkarte

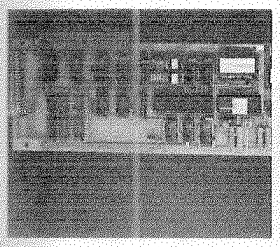




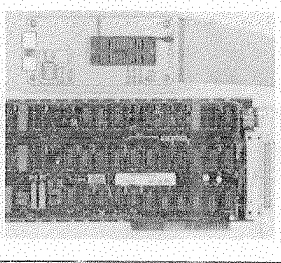




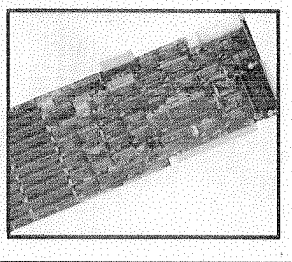
Typ	Overachiever	PC-26	PC-30	PC-14
Hersteller bzw. Vertrieb	Macrotron, München	Meilhaus, Puchheim	Meilhaus, Puchheim	Meilhaus, Puchheim
Preis (DM)	1930	910	1700	912
Funktion	Multifunktion	A/D-Wandler	A/D-, D/A-Wandler	Digital-E/A; Zähler
Einsetzbar in	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT und Kompatible	IBM-PC/-XT und Kompatible	IBM-PC/-XT/-AT und Kompatible
Speicher	bis max. 3 MByte	–	–	–
Massenspeicher	–	–	–	–
Grafik	–	–	–	–
A/D, D/A	–	16-Kanal-12-Bit-A/D-Wandler	16-Kan.-12-Bit-A/D-, 4-Kan.-12-Bit-D/A-Wandler	–
Uhr/Kalender	●	–	–	–
Schnittstellen	2 ser., 1 par., Game Port	–	–	–
Software	Dienstprogramme	Basic und Turbo-Pascal	Turbo-Pascal	Demo-Programm
Bemerkungen	Umschaltbar auf XT- und AT-Modus	Abtastraten bis 2 kHz, bipolarer od. unipolarer Betrieb, interruptfähig, extern triggerbar	Meßrate mit Standard-Software rund 2 kHz, interruptfähig	48 programmierbare E/A-Leitungen, drei 16-Bit-Zähler/Zeitgeber, Zählrate bis 2 MHz



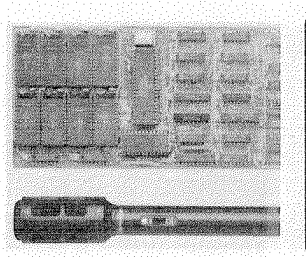
PCEFAC1
Konplan, Mannheim
rd. 1350
Meßwerterfassung
IBM-PC/-XT/-AT und Kompatible
–
–
–
16-Kanal-AC/DC-Effektiv-Wertmessung
–
–
–
Effektivwertmessungen von 0...3 V bei 0...300 kHz, Meßdauer 1 s/Kanal



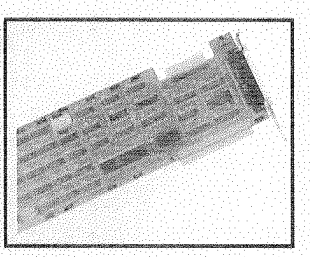
PC Sprint Prog
List & Niemann, Wiesbaden
3306
PALs und EPROMs
IBM-PC/-XT/-AT und Kompatible
–
–
–
–
–
Programmieradapter
PLD-Assembler
Externer Adapter mit der Programmierfassung, kompatibel mit Abel, Cupl, Palasm



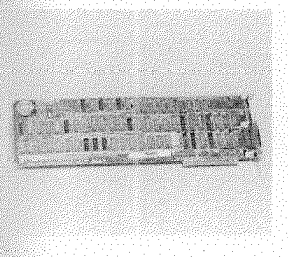
7-Pack
Macrotron, München
784
Multifunktion
IBM-PC/-XT/-AT und Kompatible
0...384 KByte
–
–
–
–
ser., par., Game Port
RAM-Disk, Spooler
Multifunktionskarte mit Speichererweiterung bis 384 KByte (1559 DM)



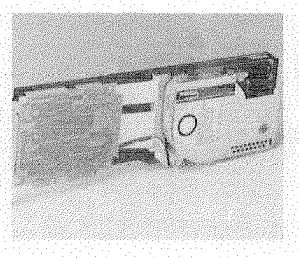
Little Dictator
Macrotron, München
1995
Spracheingabe
IBM-XT/-AT und Kompatible
–
–
–
–
–
–
inkl.
Spracheingabe bis 500 Befehlsworte, benötigt 64 KByte Speicher, inkl. Mikrofon



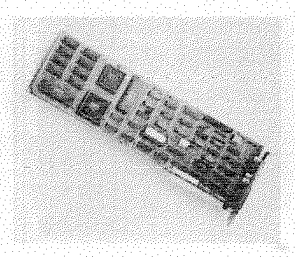
Drive Master
Macrotron, München
781
Controller
IBM-PC/-XT und Kompatible
–
–
–
–
–
–
–
Steuerkarte für 4 Plattenlaufwerke gleichzeitig, gemischter Betrieb von 3½, 5½ und 8"



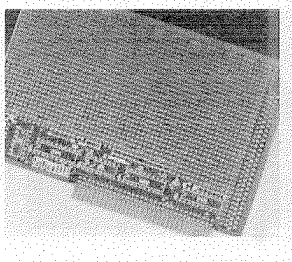
GPIP PCIII
National Instruments, Vertrieb Meilhaus, Puchheim
3397
IEEE-488-Schnittstelle
IBM-XT/-AT und Kompatible
–
–
–
optional
IEEE 488
Treiberprogramm
Verbindet PCs mit Meßgeräten mit IEEE-Schnittstelle, mit Echtzeituhr 3853 DM



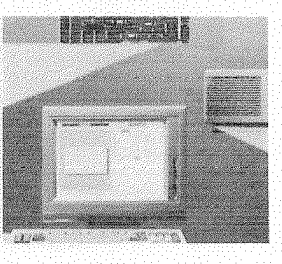
Megacard 20
Merlin, Tittmoning
1908
Festplattenkarte
IBM-PC/-XT und Kompatible, Olivetti M21 u. M24
–
20 MByte
–
–
–
Firmware
Megacard 32 mit 32-MByte-Speicher für 2070 DM



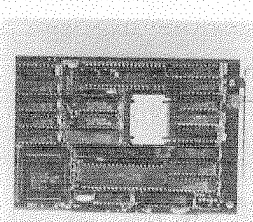
Sega
MF-Tronic, Heimstetten
k. A.
Grafik
IBM-PC/-XT/-AT und Kompatible
256 KByte Bildspeicher
–
EGA-, CGA-Standard
–
–
Centronics
–
Grafikadapter für Farb- und Monochrom-Grafiken

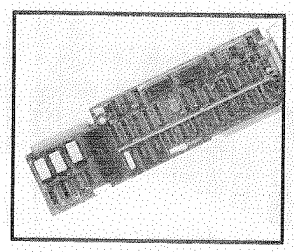


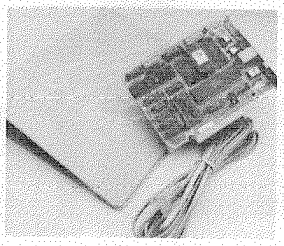
Shorty
µTronic, München
98
Prototypen-Karte
IBM-PC/-XT und Kompatible
–
–
–
–
–
Volle 64-KByte-E/A-Adreßcodierung
–
Für DMA-Betrieb ausgelegt, Adressen in 16-, 32- oder 46er Schritten einstellbar

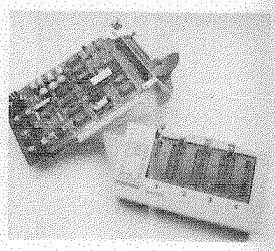


NDC 286
Nokia, Starnberg
3020
Grafik
IBM-AT und Kompatible
128-KByte-Bildspeicher
–
Emuliert Hercules und CGA (4 Graustufen)
–
–
–
Schrift wird auf Nokia-Bildschirm schwarz auf weiß dargestellt; entspricht den Anforderungen der Berufsgenossenschaften

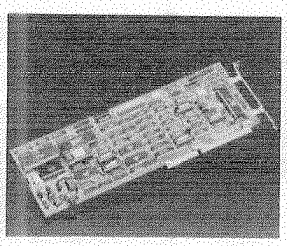


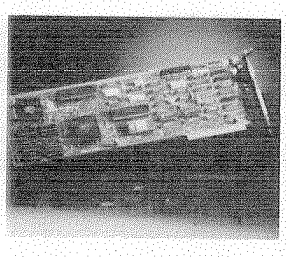


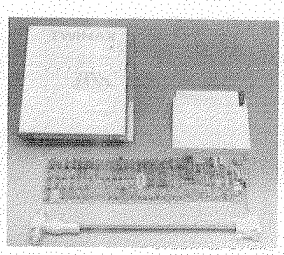


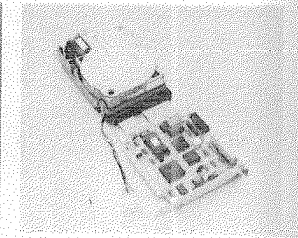


Typ	I/119	I/137S Irma	I/LAN	I/123 EPROM-Burner
Hersteller bzw. Vertrieb	Pyramid, Freiburg	Pyramid, Freiburg	Pyramid, Freiburg	Pyramid, Freiburg
Preis (DM)	1100	3306	2000	490
Funktion	Beschleunigerkarte	Schnittstellen	Netzwerk	Sonstige
Einsetzbar in	IBM-PC/-XT und Kompatible	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT und Kompatible	IBM-PC/-XT/-AT und Kompatible
Speicher	–	–	–	–
Massenspeicher		–	–	–
Grafik	–	–	–	–
A/D, D/A	–	–	–	–
Uhr/Kalender	–	–	–	–
Schnittstellen	–	IBM-3270-Emulation	–	Programmieradapter
Software	–	Treiber	inkl.	inkl.
Bemerkungen	Beschleunigerkarte mit 80286	Verbindet PC mit Großrechnern	Netzwerkkarte für bis zu 255 Teilnehmer, 1,5 km Entfernung, Starterkit mit 2 Adapterkarten	Programmiert bis 4 EPROMs bis zu 512 KByte gleichzeitig

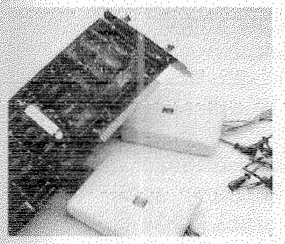




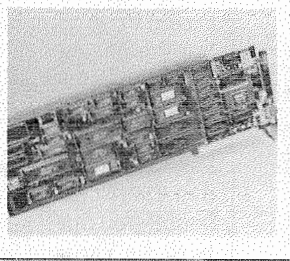




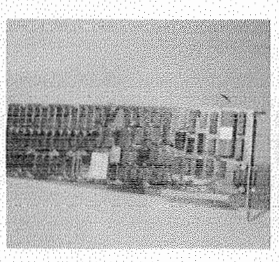
Typ	6620	8851 E	Twinaxis FT Plus	Vortex-Card
Hersteller bzw. Vertrieb	LCS/Unitronic, Düsseldorf	LCS/Unitronic, Düsseldorf	Vector, Köln	Vortex, Flein
Preis (DM)	582	635	k. A.	1398
Funktion	Controller	Grafik	Emulator	Festplattenkarte
Einsetzbar in	IBM-AT und Kompatible	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT/-AT und Kompatible	IBM-PC/-XT/-AT und Kompatible
Speicher	–	256-KByte-Bildspeicher	–	–
Massenspeicher	–	–	–	20 MByte
Grafik	–	EGA-, CGA-, MDA-kompatibel	–	–
A/D, D/A	–	–	–	–
Uhr/Kalender	–	–	–	–
Schnittstellen	ST 506, ST 412	–	Twinax-Schnittstelle zu IBM-System 34, 36 und 38	–
Software	–	–	–	–
Bemerkungen	2 Festplatten und 2 Floppylaufwerke anschließbar	Grafikadapter zur Aufrüstung von PCs mit hochauflösenden Farbmonitoren	Verbindet PCs mit dem Großrechner, 3270-Emulation	Insbesondere für Schneider PC 1512



I/145
Pyramid, Freiburg
1499
Logik-Analysator
IBM-PC/-XT/-AT und Kompatible
–
–
–
–
–
–
inkl.
Intern: 30 MHz, 8 Kanäle und 15 MHz, 16 Kanäle; extern: 15 MHz, 16 Kanäle

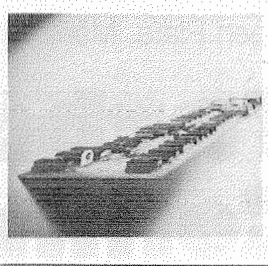


SP 100
Stac, Düsseldorf
3876
FFT-Spektrumanalysator
IBM-PC/-XT/-AT und Kompatible
–
–
–
–
–
–
Firmware inkl.
Analysiert Signale bis 12 kHz, Auflösung: 512 Frequenzlinien aus 1024 Abtastwerten

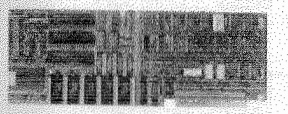


Artist 2
Techex, Taufkirchen
2804
Grafik
IBM-PC/-XT/-AT und Kompatible
128-KByte-Bildspeicher
–
640 × 400 Punkte, 16 Farben aus 4096
–
–
–
–
Controller mit NEC 7220AD, Analog-Version: 24-MHz-Bandbreite, TTL-Version 2314 DM

Revolution 512×8
Techex, Taufkirchen
5415
Grafik
IBM-PC/-XT/-AT und Kompatible
256-KByte-Bildspeicher
–
512 × 512 Punkte, 8 Bit
–
–
Analog-RGB, RS 170A
–
Mit Farbplatte: 256 Farben aus 16,8 Mio. Farben gleichzeitig darstellbar

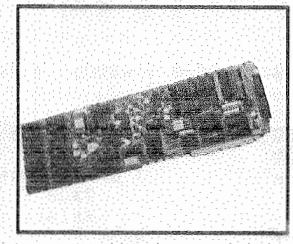


Software Decoder
Blaupunkt/Unitronic, Düsseldorf
1137
Btx-Decoder
IBM-PC/-XT/-AT, M24, T1100, Multitech, Popular
–
–
–
–
–
Modem DBT 03/1200S, RGB, V.24
Basis-Dialog-Software
CEPT-Standard, 256 KByte RAM und Modem erforderlich

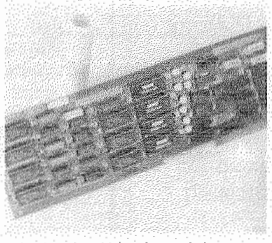


PC-FFT
Wacker, Niedenau
6099
Fourierprozessorkarte
IBM-PC/-XT/-AT und Kompatible
–
–
–
–
–
–
inkl.
Schnelle und inverse Fouriertransformation, spektrale Leistungsdichte, Gewichtsfunktion, bis zu 2048 Stützstellen

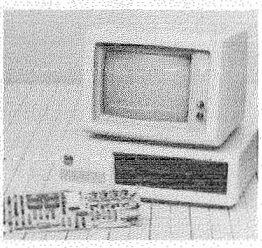
DSP 16
Wacker, Niedenau
9564
Datenakquisitions-Prozessor
IBM-PC/-XT/-AT und Kompatible
32 KByte für TMS 32020
–
–
–
–
–
inkl.
Abtastrate max. 50 kHz, min. 5 kHz, 0...16 kHz Frequenzbereich



ISC-16
Ziegler, Mönchengladbach
10824
12-Bit-A/D-Wandler
IBM-PC/-XT/-AT und Kompatible
64 KByte
–
–
–
A/D-Wandler (4...16 Meßkanäle)
–
–
inkl.
Zur schnellen Meßwerterfassung mit Abtastraten bis zu 1 Mio. Messungen/s bei 12 Bit Auflösung



RC-204
Ziegler, Mönchengladbach
5124
Puls/Funktionsgen.
IBM-PC/-XT/-AT und Kompatible
–
–
–
–
4-Kanal-12-Bit-D/A-Wandler
–
–
inkl.
4 programmierbare Generatoren



BE 485
Ziegler, Mönchengladbach
5694
Transientenrecorder
IBM-PC/-XT/-AT und Kompatible
16 KByte
–
–
–
–
–
–
inkl.
4 Meßkanäle, 50000 Messungen/s, 12 Bit Auflösung

Jürgen Plate, Manfred Luckmann

Turbo ruft Command

2. Teil

In mc 11/86 wurde gezeigt, wie man aus einem Turbo-Pascal-Programm heraus DOS-Kommandos aufrufen kann – mit einer DOS-Funktion, die schon manchen Programmierer fast in den Wahnsinn getrieben hat, da die Funktion $4B erst ab der Version 3.1 des Betriebssystems MS-DOS richtig funktioniert.

Wir haben daher zwei Programme geschrieben, eines für die älteren DOS-Versionen und eines mit der Funktion $4B des Interrupt $21, eine leichte Abwandlung des obengenannten Artikels.

Auf die Version für DOS 2.11 hat uns ein Artikel im PC-Magazin gebracht („Der EXEC-Trick", Heft 18/1986). Dabei wird ein undokumentierter Interrupt mit der Nummer $2E benutzt, der vom DOS selbst verwendet wird – denn schließlich muß der Befehlsinterpreter Command.Com ja auch Programme ausführen können. Solche aus einer Applikation heraus aufgerufenen Programme nennt man übrigens „child process".

Der Interrupt $2E

Der EXEC-Aufruf mit dem „illegalen" Interrupt $2E arbeitet natürlich nicht nur bei DOS 2.11, sondern auch bei jüngeren Versionen. Es besteht jedoch die Gefahr, daß diese Funktion eines Tages ersatzlos gestrichen wird. Die Parameterversorgung ist recht simpel: Das Register SI zeigt auf eine Kommandozeile, deren erster Buchstabe ignoriert wird und die mit einem Return ($0D) abgeschlossen sein muß. Der Pascal-Typ String paßt also perfekt. Damit das Turbo-Programm nach dem EXEC-Aufruf nicht verrückt spielt, müssen nicht nur alle Register, sondern auch Stacksegment und Stackpointer gerettet werden.

Sehen Sie sich dazu die Funktion Exec im Programm aus *Bild 1* an. Als Parameter werden an die Funktion erstens die Kommandozeile und zweitens die Länge des laufenden Programms (aufgerundet) übergeben. Die Programmlänge erfährt man beim Compilieren. Man setzt am Schluß die Konstante Program_Size auf einen passend aufgerundeten Wert.

Da ein Programm in einer Einplatzumgebung zunächst den ganzen Speicher zugeteilt bekommt, wäre beim Exec-Aufruf für den Kind-Prozeß kein Speicher frei.

Die Exec-Funktion gibt also zunächst den nicht vom Vater-Programm selbst benötigten Speicher zurück (und dazu wird die Variable Size benötigt). Das geschieht zu Beginn des Anweisungsteils mit der Funktion $4A des Interrupt $21.

Anschließend werden alle Register auf den Stack gerettet und der Stackpointer sowie das Stacksegment in den initialisierten Konstanten SVSP und SVSS gespeichert. Danach wird DS=SS gesetzt und die Adresse der Kommandozeile nach SI gebracht (das Ganze geht natürlich nur als Maschinenprogramm in Inline-Code). Nun wird der Interrupt $2E aufgerufen. Der Exec-Aufruf setzt im Fehlerfall das Carry-Flag. Die Flags werden für spätere Untersuchungen in die Variable Flags übertragen. Danach werden Stacksegment, Stackpointer und Register restauriert und das Vater-Programm läuft weiter. Trat ein Fehler beim

```
Program Test;
{ Achtung ! Compilieren mit:
  cOde >= Program_Size, mInimum stack = mAximum stack = 400 }

Const
  Program_Size = $400;  { je nach Programmgröße festlegen}

Type
  long_string = String[255];
  TReg        = record
                  case Boolean of
                    True : (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags:Integer);
                    False: (AL,AH,BL,BH,CL,CH,DL,DH:Byte);
                end;

Var
  s :long_string;
  dummy : Boolean;

Function Exec(Co : Long_string; size: integer): Boolean;
const SVSS : Integer = 0;
      SVSP : Integer = 0;
      Flags: Integer = 0;
var   Regs : TReg;

begin
CO := CO+#13;
{ Speicher freimachen }
Regs.AX := $4A00;
Regs.ES := Cseg;
Regs.BX := Sseg-Cseg+Program_size;
MSDOS(Regs);
if (Regs.flags and 1)=1 then
  begin exec := false; exit end;
INLINE(
{ Register retten }
$50/                { PUSH    AX            }
$53/                { PUSH    BX            }
$51/                { PUSH    CX            }
$52/                { PUSH    DX            }
$1E/                { PUSH    DS            }
$06/                { PUSH    ES            }
$56/                { PUSH    SI            }
```

Bild 1. Exec-Funktion mit Int $2E für DOS ab Version 2.11

```
                    {  PUSH    DI              }
                    {  PUSH    BP              }
                    {  PUSHF                   }
  Stack retten }
                    {  CLI                     }
                    {  CS:                     }
$16/SVSS/           {  MOV     [SVSS],SS       }
                    {  CS:                     }
$26/SVSP/           {  MOV     [SVSP],SP       }
                    {  STI                     }
  EXEC-Aufruf }
$D0/                {  MOV     AX,SS           }
$D8/                {  MOV     DS,AX           }
$76/<CO/            {  LEA     SI,[BP+CO]      }
$2E/                {  INT     2E              }
  Flags holen }
                    {  PUSHF                   }
                    {  POP     AX (= Flags)    }
$A3/Flags/          {  MOV     [CS:Flags],AX   }
  Alles wieder restaurieren }
                    {  CLI                     }
                    {  CS:                     }
$16/SVSS/           {  MOV     SS,[SVSS]       }
                    {  CS:                     }
$26/SVSP/           {  MOV     SP,[SVSP]       }
                    {  STI                     }
                    {  POPF                    }
                    {  POP     BP              }
                    {  POP     DI              }
                    {  POP     SI              }
                    {  POP     ES              }
                    {  POP     DS              }
                    {  POP     DX              }
                    {  POP     CX              }
                    {  POP     BX              }
                    {  POP     AX              }
exec := not ((Flags and 1)=1);
end;


begin
repeat
 write('Command: ');
 readln(s);
 if s<>'' then  Dummy := exec(s,Program_Size);
until (s='') or not Dummy;
end.
```

Aufruf des Kind-Prozesses auf, liefert die Funktion den Wert FALSE zurück.

Das Hauptprogramm in Bild 1 fordert zum Testen so lange Kommandos zur Ausführung über die Exec-Funktion an, bis ein Fehler auftritt oder man nur ein leeres Return eingibt.

Die Funktion $4B des Interrupt $21

Hier ist das Leben nicht ganz so leicht, grundsätzlich läuft es aber nach dem gleichen Schema wie oben beschrieben ab.

Die Exec-Funktion $4B hat, wie schon in mc 11 geschildert, eine recht umfangreiche Parameterversorgung (siehe Kommentar in *Bild 2*). Es kann festgelegt werden, welches Programm aufgerufen werden soll, es wird ein Suchpfad (PATH-Kommando) mitgegeben, zwei File-Control-Blöcke für Dateiparameter werden mitgegeben und ein eigenes Environment angelegt. Im Prinzip läuft ge-

```
Program Test;
{ Achtung ! Compilieren mit:
  cOde >= Program_Size, mInimum stack = mAximum stack = 400

Parameter für EXEC:
     DS:  Segment (Pfadname)
     DX:  Offset (Pfadname)
     ES:  Segment (Parameter-Block)
     BX:  Offset (Parameter-Block)
     AX:  $4B00


Exec-Parameter-Block:
     +-------------------------+
     | Environment-Segment     |
     +-------------------------+-------------------------+
     | Kommandozeile (Ofs)     | Kommandozeile (Seg)     |
     +-------------------------+-------------------------+
     | FCB 1 (Ofs)             | FCB 1 (Seg)             |
     +-------------------------+-------------------------+
     | FCB 2 (Ofs)             | FCB 2 (Seg)             |
     +-------------------------+-------------------------+     }

Const
  Program_Size = $400;  { je nach Programmgröße }

Type
  long_string = String[255];

Var
  s :long_string;
  dummy : Boolean;

Function Exec(command : Long_string; size: integer): Boolean;

const
  SVSS : Integer = 0;
  SVSP : Integer = 0;

type
  string80    = string[80];
  TReg        = record
                  case Boolean of
                    True : (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags:Integer);
                    False: (AL,AH,BL,BH,CL,CH,DL,DH:Byte);
                end;
  Dummy_FCB   = array [0..36] of Byte;
  Env_Pack    = record
                  env : Integer;
                  cmdl : ^string80;
                  fcb1, fcb2 : ^Dummy_FCB;
                end;

var
  name : String[32];
  cmd  : string80;
  FCB1,
  FCB2 : Dummy_FCB;
  paket: Env_pack;
  regs : Treg;

  function getenvironment (s: string80): string80;
    type
      arr80 = array [1..80] of char;
      cptr  = ^arr80;
    var
      p1 : cptr;
      envseg : integer absolute cseg:$2C;
      l : integer;
      found : boolean;
    begin
      p1 := ptr(envseg,0); found := false; l := length(s)+1;
      while (not found) and (p1^[1]<>#0) do
        begin
        found := (s+'=')=(copy(p1^,1,l));
        if not found then
          p1 := ptr(seg(p1^),ofs(p1^)+pos(#0,p1^));
        end;
      if found then
        begin
        p1 := ptr(seg(p1^),ofs(p1^)+1);
        getenvironment := copy(p1^,1,pos(#0,p1^)-1);
        end
```

Bild 2. Exec-Program für DOS ab Version 3.0

nau das gleiche Verfahren wie bei einem Kommandoaufruf ab. Das Kommando wird im voreingestellten oder angegebenen Pfad gesucht. Ist das Programm gefunden, wird ein 256 Byte langer Vorspann (PSP = Program Segment Prefix) mit einer Kopie der Kommandozeile und zwei Dateikontrollblöcken sowie einem Environment angelegt. Oberhalb des PSP wird dann das Programm geladen. Das DOS reserviert dabei den restlichen zur Verfügung stehenden Speicher für das Programm (siehe oben). Nach dem Ende des Programms wird die Kontrolle wieder an Command.Com zurückgegeben. Man muß also für jeden Programmaufruf die in Bild 2 gezeigten Parameter von der Exec-Funktion aus übergeben – nicht gerade einfach.

```
      else
        getenvironment :='';
    end;

begin
{ Speicher freimachen }
Regs.AX := $4A00;
Regs.ES := Cseg;
Regs.BX := Sseg-Cseg+Program_size;
MSDOS(Regs);
if (Regs.flags and 1)=1 then
  begin exec := false; exit end;

Inline(                 { Register retten }
$50/                    {  PUSH    AX             }
$53/                    {  PUSH    BX             }
$51/                    {  PUSH    CX             }
$52/                    {  PUSH    DX             }
$1E/                    {  PUSH    DS             }
$06/                    {  PUSH    ES             }
$56/                    {  PUSH    SI             }
$57/                    {  PUSH    DI             }
$55/                    {  PUSH    BP             }
$9C/                    {  PUSHF                  }
{ Stack retten }
$FA/                    {  CLI                    }
$2E/                    {  CS:                    }
$8C/$16/SVSS/           {  MOV     [SVSS],SS      }
$2E/                    {  CS:                    }
$89/$26/SVSP/           {  MOV     [SVSP],SP      }
$FB);                   {  STI                    }

{ exec-Aufruf }
name := getenvironment('COMSPEC')+#0;
Regs.DS := Seg(Name);
Regs.DX := Ofs(name)+1;
cmd := '/C '+command+#13;
paket.fcb1 := addr(FCB1);
paket.fcb2 := addr(FCB2);
paket.env := 0;
paket.cmdl := Addr(cmd);
Regs.ES := Seg(paket);
Regs.BX := Ofs(paket);
Regs.AL := 0;
Regs.AH := $4B;
MSDOS(Regs);
exec := not ((regs.Flags and 1)=1);

Inline(                 { Alles wieder restaurieren }
$FA/                    {  CLI                    }
$2E/                    {  CS:                    }
$8E/$16/SVSS/           {  MOV     SS,[SVSS]      }
$2E/                    {  CS:                    }
$8B/$26/SVSP/           {  MOV     SP,[SVSP]      }
$FB/                    {  STI                    }
$9D/                    {  POPF                   }
$5D/                    {  POP     BP             }
$5F/                    {  POP     DI             }
$5E/                    {  POP     SI             }
$07/                    {  POP     ES             }
$1F/                    {  POP     DS             }
$5A/                    {  POP     DX             }
$59/                    {  POP     CX             }
$5B/                    {  POP     BX             }
$58);                   {  POP     AX             }
end;


begin
repeat
 write('Command: ');
 readln(s);
 if s<>'' then  dummy := exec(s,Program_Size);
until (s='') or not dummy;
end.
```

Die ganze Sache wird sehr viel einfacher, wenn man sich erinnert, daß es einen Spezialaufruf des Command.Com gibt: COMMAND /C <Kommando>. Dabei wird der Command.Com geladen und führt danach automatisch den Aufruf des Kommandostrings <Kommando> durch. Die Belegung von Environment und FCBs kann man vergessen.

Genau das macht die Exec-Funktion in Bild 2. Der Aufruf der Exec-Funktion hat die gleichen Parameter wie bei jenem von Bild 1. Der Parameterblock wird als Record Env-Pack definiert. Um auf jeden Fall sicherzugehen, werden noch zwei Felder mit 36 Byte Länge als FCBs definiert.

Das Freimachen des Speichers erfolgt genau wie in Bild 1 über die Int-$21-Funktion $4A. Anschließend werden wieder die Register und der Stack gerettet. Nun muß man nachsehen, wo sich im System der Command.Com befindet.

Dazu sucht die Funktion Getenvironment nach der COMSPEC-Definition, die von der Form „COMSPEC=<Pfad für Command.Com>" ist. Die Adresse des so gefundenen Pfads wird in DS und DX übergeben. Da in Turbo die erste Feldkomponente die Länge des Strings enthält, muß der Offset um 1 erhöht werden. Anschließend wird der Parameterblock mit Kommandozeile und Dummywerten gefüllt. Jetzt kann die Exec-Funktion über den Interrupt $21 aufgerufen werden.

Diese lädt den Command.Com und führt das Kommando aus – ganz egal, ob es sich um ein internes Kommando, eine Batchdatei, eine *.EXEC-Datei oder eine *.COM.Datei handelt. Nach der Rückkehr vom Exec-Interrupt werden der Turbo-Stack und die Register wieder restauriert und die Funktion gibt den Wert der Flags zurück.

Wir haben die Programme auf verschiedenen Rechnern laufen lassen (PC, PC/XT, Olivetti M24, AT) und hatten keine Probleme. Sollte bei Ihnen etwas nicht so klappen, sind wir natürlich daran interessiert, Ihre Informationen in der mc weiterzugeben.

Andreas Töreki, Michael Reddehase

Grafik für Joyce

Wer die grafischen Fähigkeiten des Joyce nutzen wollte, suchte bislang vergebens nach der Dokumentation und Anleitung für die mitgelieferte Grafikschnittstelle GSX. Das hier vorgestellte Programm in Turbo-Pascal setzt diesem Mangel ein Ende.

Tragender Gedanke bei der Entwicklung von CP/M war es, ein geräteunabhängiges Betriebssystem zu schaffen, das Kompatibilität von Anwenderprogrammen systemübergreifend gewährleistet. Eines konnte CP/M allerdings nicht: Grafik unterstützen. So entwickelte Digital Research GSX, eine CP/M-Erweiterung, um auch Grafik systemunabhängig unterstützen zu können.
Philosophie von GSX ist es, Grafiken unabhängig von den Eigenschaften des gewählten Ausgabegerätes, wie Bildschirm, Drucker oder Plotter zu erstellen. So muß man sich nicht um die Auflösung oder andere Parameter des jeweiligen Ausgabegerätes kümmern, sondern teilt dem Computer einfach mit, auf welche Ausgabeeinheit die Grafik ausgegeben werden soll, den Rest besorgt GSX.
GSX arbeitet deshalb mit einem relativen Koordinatensystem. Mit Werten von 0 bis 32767 für die X-, wie auch Y-Achse. Der Ursprung liegt mit 0,0 in der linken unteren Ecke des Bildschirms.
Diese relativen Koordinaten werden von GSX in gerätespezifische Koordinaten umgesetzt, für Joyce: Bildschirm 720 × 256, Drucker 480 × 672 (geringe Auflösung) sowie 960 × 1368 (hohe Auflösung), Plotter HP 7470A (spezifisch). Die für GSX benötigten Geräteinformationen befinden sich in den Gerätetreiberdateien. Diese sind in der Datei „ASSIGN .SYS" mit einer zugewiesenen Benutzernummer aufzuzählen (*Bild 1*).
Um nun in den Genuß der Fähigkeiten von GSX-Grafiken zu kommen, bindet man das Listing (*Bild 2*) als Include-Datei in eigene Programme ein. Dann braucht man ein .COM File des Programmes. Dies erreicht man durch die Compiler-Option COM. Wichtiger Hinweis: Hier sollte die Endadresse auf $B600 festgelegt werden, um Platz für Gerätetreiber zu reservieren, da es ansonsten beim Einbinden von GSX unweigerlich die Fehlermeldung „Not enough memory for GSX" geben würde.
Schließlich muß man dem compilierten Programm noch GSX einbinden; dies wird mit Hilfe von GENGRAF.COM vollzogen, indem man GENGRAF ⟨Name des .COM Files⟩ aufruft. GSX ist danach in dieses .COM File für immer eingebunden.

```
22 @:DDFXHR8
21 @:DDFXLR8
01 @:DDSCREEN
```

Für den CPC 6128 muß Assign.sys geändert werden. z.B.:

```
21 @:DDFXLR7
01 @:DDMODE2
```

Bild 1. ASSIGN.SYS

Funktionsweise von GSX

GSX besitzt einen Parameterblock, in dem Steuercodes übergeben und dann von GSX interpretiert werden (z. B. Linie, Koordinatenpaare, usw.). Die GSX-Routine wird mit dem BDOS-Aufruf 115 aktiviert, zugleich muß im DE-Register diejenige Adresse stehen, bei der der Parameterblock beginnt. Unter Turbo-Pascal wurde dieser Parameterblock durch Arrays realisiert, was den Vorteil eines leichten Zugriffes mit sich bringt. Auf die Steuercodes kann an dieser Stelle nicht eingegangen werden, da sie den Rahmen des Artikels überschreiten würden (s. Literaturhinweis).

Literatur:

GSX User's Guide/Programmer's Guide, Digital Research, Markt & Technik Verlag.

Erweiterte Grafikprozeduren

GRAFIC_ON (⟨Device⟩):	Mit diesem Befehl wird der Grafik-Modus eingeschaltet und das Ausgabegerät festgelegt.
⟨Device⟩:	⟨Monitor⟩ Bildschirm als Ausgabegerät ⟨Printer_LR⟩ Drucker mit geringer Auflösung ⟨Printer_HR⟩ Drucker mit hoher Auflösung ⟨Plotter⟩ Plotter HP7470A
GRAFIC_OFF:	Schaltet den Grafik-Modus ab. Sofern das Ausgabegerät ein Drucker ist, wird ein Ausdruck der Grafik erstellt.
CLR_GRAFIC:	Löscht den Grafik-Bildschirm
PLOT (⟨X⟩,⟨Y⟩):	Setzt einen Punkt
⟨X⟩:	⟨0...32767⟩
⟨Y⟩:	⟨0...32767⟩
Line (X1,Y1,X2,Y2):	Zieht eine Linie von (X1, Y1) nach (X2, Y2)
⟨X1,Y1,X2, Y2⟩:	⟨0...32767⟩
BOX (X1,Y1,X2,Y2):	Zeichnet Rechteck mit linker unterer Ecke X1,Y1 und rechter oberer Ecke X2,Y2
⟨X1,Y1,X2,Y2⟩:	⟨0...32767⟩
PRINTXY (X,Y,⟨TEXT⟩):	Schreibt Text an Stelle ⟨X,Y⟩
⟨X,Y⟩:	⟨0...32767⟩
⟨Text⟩:	String mit max. Länge 10
COLOR (⟨Nummer⟩):	Farbe der Punkte, Linien, Boxen
⟨Nummer⟩:	⟨0⟩ Weiß
⟨Nummer⟩:	⟨1⟩ Schwarz
LINE_TYPE (⟨Nummer⟩):	legt den Linientyp fest (s. JOYCE CP/M Benutzerhandbuch)
SET_CHARHEIGHT (⟨Nummer⟩):	Gilt nur für Drucker/Plotter. Legt Schriftgröße fest.
⟨Nummer⟩:	⟨1...12⟩ klein...groß
DIRECTION (⟨Nummer⟩):	Gilt nur für Drucker/Plotter. Legt Druckrichtung vom Text, entgegen dem Uhrzeigersinn, fest.
⟨Nummer⟩:	⟨0...3⟩
FILL_TYPE (⟨Sort⟩, ⟨Option⟩):	Gilt nur für Drucker/Plotter. Legt inneres Muster der BOX fest.
⟨Sort⟩:	⟨0⟩ weiß
⟨Sort⟩:	⟨1⟩ schwarz
⟨Sort⟩:	⟨2⟩ getönt ⟨Option = 1...6⟩ (s. JOYCE CP/M-Benutzerhandbuch)
⟨Sort⟩:	⟨3⟩ schraffiert ⟨Option = 1...6⟩ (s. JOYCE CP/M-Benutzerhandbuch)

```
CONST     Gsx             = 115;
          Monitor_ID      = 01;
          Printer_LR_ID   = 21;
          Printer_HR_ID   = 22;
          Plotter_ID      = 11;

TYPE      String_10       = STRING [10];
          DevType         = (Monitor,Printer_LR,Printer_HR,Plotter);

VAR       DE_Address,
          X_Resolution,
          Y_Resolution    : Integer;
          Pb              : Array [1..5]  of Integer;
          Contrl          : Array [1..6]  of Integer;
          Intin           : Array [1..10] of Integer;
          Intout          : Array [1..45] of Integer;
          Ptsin           : Array [1..4]  of Integer;
          Ptsout          : Array [1..12] of Integer;

PROCEDURE Grafic_On (Dev: Devtype);
VAR       I : Integer;
BEGIN
  Write (Chr(27),'0');          { Statuszeile  ausblenden }
  Write (Chr(27),'y');          { 32 * 90 Bildschirmmodus }
  DE_Address  := Addr(Pb [1]);
  Pb[1]       := Addr(Contrl [1]);
  Pb[2]       := Addr(Intin [1]);
  Pb[3]       := Addr(Ptsin [1]);
  Pb[4]       := Addr(Intout [1]);
  Pb[5]       := Addr(Ptsout [1]);
  Contrl [1] := 1;
  Contrl [2] := 0;
  Contrl [4] := 10;
  CASE Dev OF
    Monitor    : Intin [1] := Monitor_ID;
    Printer_LR : Intin [1] := Printer_LR_ID;
    Printer_HR : Intin [1] := Printer_HR_ID;
    Plotter    : Intin [1] := Plotter_ID;
  END;
  FOR I := 2 TO 10 DO
    Intin [I] := 1;
  Bdos(Gsx,DE_Address);
  Contrl [1] := 5;
  Contrl [6] := 2;
  Contrl [2] := 0;
  Bdos(Gsx,DE_Address);
  X_Resolution := Intout [1];
  Y_Resolution := Intout [2];
END;

PROCEDURE Grafic_Off;
BEGIN
  Contrl [1] := 4;
  Contrl [2] := 0;
  Bdos (Gsx,DE_Address);
  Gotoxy (0,0);
  Contrl [1] := 5;
  Contrl [6] := 3;
  Contrl [2] := 0;
  Bdos(Gsx,DE_Address);
  Contrl [1] := 2;
  Contrl [2] := 0;
  Bdos(Gsx,DE_Address);
  Write (Chr(27),'1');
  Write (Chr(27),'e');   { Cursor sichtbar }
END;

PROCEDURE Clr_Grafic;
BEGIN
  Contrl [1] := 3;
  Contrl [2] := 0;
  Bdos(Gsx,DE_Address);
END;

PROCEDURE DIRECTION (I: INTEGER);
BEGIN
  Contrl [1] := 13;
  Contrl [2] := 0;
  Intin  [1] := I * 900;
  Intin  [2] := Round (100 * Cos (I * Pi / 2));
  Intin  [3] := Round (100 * Sin (I * pi / 2));
  Bdos (Gsx, DE_Address);
END;
```

```
PROCEDURE Color (Typ: Integer);
BEGIN
  Contrl [1] := 17;
  Contrl [2] := 0;
  Intin  [1] := Typ;
  Bdos(Gsx,DE_Address);
END;

PROCEDURE Set_CharHeight(Height: Integer);
BEGIN
  Contrl [1] := 12;
  Contrl [2] := 1;
  Contrl [3] := 2;
  Ptsin  [1] := 0;
  Ptsin  [2] := Height*192;
  Bdos (Gsx, DE_Address);
END;

PROCEDURE Line_Type (Typ: Integer);
BEGIN
  Contrl [1] := 15;
  Contrl [2] := 0;
  Intin  [1] := Typ;
  Bdos(Gsx,DE_Address);
END;

PROCEDURE Fill_Type (Sort,Option: Integer);
BEGIN
  Contrl [1] := 23;
  Contrl [2] := 0;
  Intin  [1] := Sort;
  Bdos (Gsx,DE_Address);
  Contrl [1] := 24;
  Contrl [2] := 0;
  Intin  [1] := Option;
  Bdos (Gsx, DE_Address);
END;

PROCEDURE Line (X1,Y1,X2,Y2: Integer);
BEGIN
  Contrl [1] := 6;
  Contrl [2] := 2;
  Ptsin  [1] := X1;
  Ptsin  [2] := Y1;
  Ptsin  [3] := X2;
  Ptsin  [4] := Y2;
  Bdos(Gsx,DE_Address);
END;

PROCEDURE Plot (X1,Y1: Integer);
BEGIN
  Contrl [1] := 6;
  Contrl [2] := 2;
  Ptsin  [1] := X1;
  Ptsin  [2] := Y1;
  Ptsin  [3] := X1;
  Ptsin  [4] := Y1;
  Bdos(Gsx,DE_Address);
END;

PROCEDURE Printxy (X_Pos,Y_Pos: Integer ; Text: String_10);
VAR       I : Integer;
BEGIN
  Contrl [1] := 8;
  Contrl [2] := 1;
  Contrl [4] := Length (Text);
  FOR I := 1 TO Length (Text) DO
    Intin [I] := Ord (Text [I]);
  Ptsin [1] := X_Pos;
  Ptsin [2] := Y_Pos;
  Bdos(Gsx,DE_Address);
END;

PROCEDURE Box (X1,Y1,X2,Y2: Integer);
BEGIN
  Contrl [1] := 11;
  Contrl [2] := 2;
  Contrl [6] := 1;
  Ptsin  [1] := X1;
  Ptsin  [2] := Y1;
  Ptsin  [3] := X2;
  Ptsin  [4] := Y2;
  Bdos (Gsx,DE_Address);
END;
```

Bild 2. Als Include-Datei einzubinden

Erwin Cramer

Erweiterter Bresenham-Algorithmus

Mit wenigen Zeilen in Turbo-Pascal werden Grafik-Programmen verschiedene Linienarten beigebracht. Die Unterscheidung von diagonalen, horizontalen und vertikalen Linien sorgt für eine beachtliche Beschleunigung.

Bei vielen Grafikpaketen oder Grafikspracherweiterungen können beim Zeichnen verschiedene Linienarten eingesetzt werden. Durch unterschiedlich gestrichelte Linien lassen sich Umrisse von Mittellinien oder gedachten Linien unterscheiden, um so die Übersichtlichkeit von Zeichnungen zu steigern. Bei Grafikprogrammen und -erweiterungen, die nicht über diese Möglichkeit verfügen, ist diese jedoch recht einfach nachzurüsten. So auch in dem Grafikprogramm zu dem Artikel „Hochauflösende Druckergrafik" von Jürgen Plate, erschienen in mc 9/86 auf den Seiten 62...65. Und damit auch in der geschwindigkeitsmäßig aufgemöbelten Version von Horst Kling, erschienen als Leserbrief in mc 11/86, Seite 12.

Die beiden Listings zeigen, wie die Erweiterungen jeweils eingebaut wurden. Die Parameterliste der Prozedur „line" (*Bild 1*) wurde um die Variable „muster" vom Typ „set of 0...15" ergänzt, mit der das Linienmuster übergeben wird. In dieser Menge – die man sich auch als „array (0...15) of boolean" vorstellen kann, sind die Bits so gesetzt, wie nachher auch die Punkte in der Linie erscheinen sollen. Wären alle Bits gesetzt, so hätte man wieder die durchgezogene Linie; wäre die Menge leer, würde überhaupt keine Linie erscheinen; enthielte die Menge z. B. nur die geraden Zahlen, so würde nur jeder zweite Punkt gezeichnet. Da die Menge quasi 16 Bit lang ist, wiederholt sich auch in der Linie immer nach 16 Punkten das gleiche Muster.

Daß nur die gewünschten Punkte gezeichnet werden, erreicht man dadurch, daß man eine Integervariable „i" durch ständige Inkrementierung und Anwendung der Modulo-Funktion alle Werte von 0...15 durchlaufen läßt. Diese Inkrementierung erfolgt jeweils bevor ein Punkt gezeichnet werden soll, und die Überprüfung, ob der neue Wert der Variablen in der Mustermenge enthalten ist, gibt Aufschluß darüber, ob der Punkt gezeichnet werden soll oder schwarz bleibt. Im Listing 2 (*Bild 2*) wird für diese Überprüfung die Laufvariable „i1" verwandt, das Prinzip bleibt aber das gleiche. Als zweite Erweiterung wurde die Möglichkeit geschaffen, die Prozedur möglichst schnell wieder zu verlassen, falls Start- und Endpunkt der Linie zusammenfallen und damit das Zeichnen einer Verbindungslinie entfällt.

Durch diese Abfrage (Zeile 10) vereinfachen sich im Listing 1 auch die späteren Abfragen: Zeile 51 wurde verkürzt und Zeile 62 entfällt fast ganz.

Das Listing 2 wurde außerdem dergestalt ergänzt, daß nun zwischen diagonalen und vertikalen bzw. horizontalen Linien (Zeile 23) unterschieden wird. Der Algorithmus zum Zeichnen nichtdiagonaler Linien ist noch etwas kürzer und damit schneller als der übliche Bresenham-Algorithmus. Da vor allem im technischen Bereich sehr viele vertikale und horizontale Linien gezeichnet werden, läßt sich durch diese Unterscheidung doch noch ein Geschwindigkeitsgewinn erzielen.

```
 1 program line;
 2 type
 3    set15 = set of 0 .. 15;
 4 procedure line (x1, y1, x2, y2 : integer; muster : set15);
 5 label exit;
 6 var
 7    x, y, z, a, b, d, dx, dy, dp, dq, i : integer;
 8 begin
 9    dx := abs (x2 - x1); dy := abs (y2 - y1);
10    if (dx + dy) = 0 then goto exit;
11    x := x1; y := y1;
12    plot (x, y, 15);
13    i := 0;
14    if (dx <> 0) and (dy <> 0) then
15       begin
16          if dy <= dx then
17             begin
18                z := x2;
19                if x1 <= x2 then a := 1 else a := -1;
20                if y1 <= y2 then b := 1 else b := -1;
21                dp := dy + dy; d := dp - dx; dq := dp - (dx + dx);
22                while x <> z do
23                   begin
24                      i := (i + 1) mod 16;
25                      x := x + a;
26                      if  d < 0 then d := d + dp
27                      else
28                         begin y := y + b; d := d + dq; end;
29                      if i in muster then plot (x, y, 15);
30                   end;
31             end
32          else
33             begin
34                z := y2;
35                if y1 <= y2 then a := 1 else a := -1;
36                if x1 <= x2 then b := 1 else b := -1;
```

Bild 1. Mit der Prozedur „line" werden Linienmuster übergeben

```
 1 program line;
 2 type
 3    set15 = set of 0 .. 15;
 4
 5 procedure line (x1, y1, x2, y2 : integer; muster : set15);
 6 label exit;
 7 var x, y, z, dx, dy, dz, i1, i2 : integer;
 8 begin
 9    dx := abs (x1 - x2); dy := abs (y1 - y2);
10    if (dx + dy) = 0 then goto exit;
11    if x1 < x2 then
12       begin
13          x := x1; y := y1;
14          if y1 > y2 then z := -1 else z := 1;
15       end
16    else
17       begin
18          x := x2; y := y2;
19          if y2 > y1 then z := -1 else z := 1;
20       end;
21    if dx > dy then i2 := dx else i2 := dy;
22    plot (x, y, 15);
23    if (dx = 0) or (dy = 0) then
24       for i1 := 1 to i2 do
25          begin
26             if dx = 0 then
27                y := y + z
28             else x := x + 1;
29             if ((i1 - 1) mod 16) in muster then plot (x, y, 15);
30          end
31    else
32       begin
33          dz := i2 div 2;
34          for i1 := 1 to i2 do
35             begin
36                if dz < dx then
37                   begin dz := dz + dy; x := x + 1; end;
38                if dz >= dx then
39                   begin dz := dz - dx; y := y + z; end;
40                if ((i1 - 1) mod 16) in muster then plot (x, y, 15);
41             end;
42       end;
43 exit:
44 end;
```

Bild 2. Diese Routine zeichnet nur die gewünschten Punkte

```
37                dp := dx + dx; d := dp - dy; dq := dp - (dy + dy);
38                while y <> z do
39                   begin
40                      i := (i + 1) mod 16;
41                      y := y + a;
42                      if d < 0 then
43                         d := d + dp
44                      else
45                         begin x := x + b; d := d + dq; end;
46                      if i in muster then plot (x, y, 15);
47                   end;
48             end;
49       end
50    else
51       if (dx = 0) then
52          begin
53             z := y2;
54             if y1 <= y2 then b := 1 else b := -1;
55             repeat
56                i := (i + 1) mod 16;
57                y := y + b;
58                if (i in muster) then plot (x, y, 15);
59             until y = z;
60          end
61       else
62          begin
63             z := x2;
64             if x1 <= x2 then a := 1 else a := -1;
65             repeat
66                i := (i + 1) mod 16;
67                x := x + a;
68                if (i in muster) then plot (x, y, 15);
69             until x = z;
70          end;
71 exit;
72 end;
```

Neue 32-Bit-Prozessoren von AT & T

Nach der Vorstellung des WE32100 vor etwa 18 Monaten gibt es von AT & T jetzt bereits die dritte Generation von 32-Bit-Prozessoren. Der Chipsatz des WE32200 enthält über 200 000 Transistorfunktionen und wird in Zukunft für den Bau von sehr schnellen Computern sorgen.

Gefertigt wird die neue Mikroprozessor-Familie in 1-µm-CMOS-Technologie, die sehr hohe Taktraten erlaubt. AT & T wird die Bauteile zunächst in einer 24-MHz-Version anbieten; eine 30-MHz-Version wurde bereits angekündigt. Neben der eigentlichen CPU, die auf dem Chip 32 Register bietet und 25 Adressierungsarten kennt, sind bereits eine Speicherverwaltungseinheit (MMU 32301) und ein mathematischer Coprozessor (MAU 32206) verfügbar.

Die MMU arbeitet parallel zur CPU, wodurch Multi-User-/Multi-Tasking-Systeme und parallele Mehrprozessorensysteme realisiert werden können.

Hervorragendes Merkmal der MMU 32201 ist der integrierte Befehls- und Daten-Cache-Speicher. Er umfaßt 4 KByte und erlaubt verzögerungsfreien Zugriff bei einer Taktfrequenz bis zu 30 MHz. Sollten die 4-KByte-Cache-Speicher nicht ausreichen, können bis zu vier MMUs zusammengeschaltet werden.

Das 133-Pin-Gehäuse wird mit +5-V-Spannung versorgt und kann bis zu 4-GByte-Speicher (virtuell oder physikalisch) adressieren.

Die Leistung der MAU 32206 (Math Acceleration Unit) mit Gleitkomma-Arithmetik beträgt nach Angaben von AT & T 3,5 Mega-Whetstones bei einer Taktfrequenz von 30 MHz. Das 125-Pin-IC enthält acht 80-Bit-Register und benötigt nur eine Spannung von +5 V.

Die neue Prozessor-Familie ist anschluß- und objektcode-kompatibel zum Chipsatz WE32100. Anwendungen, die mit diesem Chipsatz entwickelt wurden, sollen ohne Probleme auch auf den leistungsfähigeren Modellen der WE32200-Familie laufen.

Nach Unterlagen von AT & T

Günther Sternberg

MS-DOS-Interrupt 24H

DOS-Fehlermeldungen mit Turbo-Pascal-Routinen auffangen

Einer der vielen Vorteile von MS-DOS gegenüber CP/M liegt in der komfortablen Fehlerbehandlung bei Ein-/Ausgabeoperationen. Jeder MS-DOS-Anwender kennt die Abfrage „Abort, Retry, Ignore" zusammen mit einer Fehlermeldung, zum Beispiel infolge eines Zugriffs auf ein nicht bereites Laufwerk. Oft erscheinen Fehlermeldung und Abfrage an der unpassendsten Bildschirmstelle und zerstören so eine Eingabemaske. Unser Autor beschreibt mehrere Wege zur Lösung dieses Problems bei Turbo-Pascal-Programmen.

Die Form der Fehlerbehandlung unter MS-DOS wird im Grafikmodus noch unangenehmer, da Fehlermeldung und Abfragetext als ein unlesbares wirres Punktmuster innerhalb der Grafik erscheinen. Diese Probleme lassen sich nur mit einer eigenen Fehlerroutine in den Griff bekommen.

Nach jeder fehlerhaften Ein-/Ausgabeoperation, dies kann ein Zugriff auf eine fehlerhafter Diskette oder die Ausgabe eines Zeichens an den nicht bereiten Drucker sein, löst MS-DOS den Interrupt 24H aus. Durch diesen Interrupt wird die Fehlerroutine aufgerufen, deren Adresse sich an der entsprechenden Stelle der Interrupt-Tabelle befindet.

Mehrere Register enthalten beim Aufruf dieses Unterprogramms Informationen über Art und Quelle des Fehlers. *Bild 1* zeigt den Inhalt des DI-Registers. Je nach Zustand von Bit 7 des AH-Registers wird die Fehlerursache unterschieden.

Bit 7 von AH = 0: fehlerhafter Laufwerkszugriff

```
0   Attempt to write on write-protected Disk
1   Unknown unit
2   Drive not ready
3   Unknown command
4   CRC error in data
5   Bad drive request structure length
6   Seek error
7   Unknown media type
8   Sector not found
9   Printer out of Paper
A   Write fault
B   Read fault
C   General failure
```

Bild 1. Das DI-Register enthält die Nummer des Fehlers

Bedeutung der restlichen Bits des AX-Registers:

Bit 0 des AH-Registers
- 0 Fehler bei Leseoperation
- 1 Fehler bei Schreiboperation

Bit 2...3 des AH-Registers
- 00 Fehler im MS-DOS-Bereich der Diskette
- 01 Fehler in der FAT der Diskette
- 10 Fehler im Directory der Diskette
- 11 Fehler im Datenbereich der Diskette

Bit 3 des AH-Registers
- 0 „Terminate" als Antwort nicht erlaubt
- 1 „Terminate" als Antwort erlaubt

Bit 4 des AH-Registers
- 0 „Retry" als Antwort nicht erlaubt
- 1 „Retry" als Antwort erlaubt

Bit 5 des AH-Registers
- 0 „Ignore" als Antwort nicht erlaubt
- 1 „Ignore" als Antwort erlaubt

Das AL-Register enthält die Nummer des Laufwerks
- 0 Laufwerk A
- 1 Laufwerk B etc.

Bit 7 von AH = 1: Kopie der FAT im Speicher fehlerhaft oder Ausgabefehler bei Peripheriegerät

In diesem Fall enthalten das BP- und das SI-Register die Adresse des Device-Headers vom fehlerverursachenden Gerät. Dieser Device-Header enthält nähere Angaben zum Gerät.

Mit diesen Informationen gilt es eine Entscheidung über die weitere Vorgehensweise zu treffen. Der Wert im AL-Register (*Bild 2*) informiert MS-DOS nach dem Rücksprung aus der Fehlerroutine über die getroffene Entscheidung. Die standardmäßige INT-24H-Fehlerroutine liefert entsprechend der Registerinhalte beim Aufruf die zugehörige Fehlermeldung und überläßt dem Benutzer die Entscheidung.

Wie läßt sich nun eine Zerstörung des Bildinhalts durch Fehlermeldung und Abfrage innerhalb eines Pascal-Programms vermeiden? Das Anwender-Programm muß dafür sorgen, daß nach dem Auftreten eines Ein-/Ausgabefehlers (Interrupt 24H) eine eigene Fehlerroutine aufgerufen wird. In dieser Routine wird zunächst der Bildschirminhalt gesichert, dann die INT-24H-Routine von MS-DOS

```
0     Fehler ignorieren
1     fehlgeschlagene Operation wiederholen
2     gesamten Programmlauf durch einen Aufruf
      des Interrupts 23 (sedizimal) abbrechen
3     nur die fehlgeschlagene Operation abbrechen
```

Bild 2. Mögliche Antworten des INT-24H-Handlers im AL-Register

```
(* ------------------------- Int24 - Handler -------------------------- *)

Type Switch = (On, Off); (* mögliche Zustände des eigenen Int24_Handlers *)

Const Int24_Ofs : Integer = 0;     (* Offset des alten Int24-Handlers       *)
      Int24_Seg : Integer = 0;     (* Segment des alten Int24-Handlers      *)
                                   (* Reihenfolge der Konstanten wichtig !! *)
      State     : Switch  = Off;   (* aktueller Zustand des Int24_Handlers  *)
      Turbo_ds  : Integer = 0;     (* Datensegment von Turbo                *)
      Extern_ds : Integer = 0;     (* Datensegment bei Int24-Aufruf         *)

Var Screen : Array [1..4000] of Byte Absolute $B000:0000; (* Grafikkarte *)
    Buffer : Array [1..4000] of Byte; (* Puffer zum Retten des Bildspeicher *)
    Old_X, Old_Y : Integer; (* Cursor-Position von Int24-Aufruf *)

Procedure Int24;

Begin
  Inline ($5D/$5D/                                  (* POP BP, POP BP        *)
          $83/$EC/$04/                              (* DEC SP,+04            *)
          $50/$53/$51/$52/$1E/$06/$56/$57/$55/$9C/  (* Register sichern      *)
          $2E/$8C/$1E/Extern_ds/                    (* DS-Register sichern   *)
          $2E/$8E/$1E/Turbo_ds);                    (* DS-Register von Turbo *)
  Move (Screen, Buffer, 4000);                      (* Bildschirm sichern    *)
  Old_X := WhereX;                                  (* Cursorposition retten *)
  Old_Y := WhereY;
  Inline ($2E/$8E/$1E/Extern_ds/                    (* DS-Register rest.     *)
          $9D/$5D/$5F/$5E/$07/$1F/$5A/$59/$5B/$58/  (* Register rest.        *)
          $FB/                                      (* STI                   *)
          $9C/                                      (* PUSHF                 *)
          $2E/$FF/$1E/Int24_Ofs/                    (* CALL FAR [Int24_Ofs]  *)
          $50/$53/$51/$52/$1E/$06/$56/$57/$55/$9C/  (* Register sichern      *)
          $2E/$8C/$1E/Extern_ds/                    (* DS-Register sichern   *)
          $2E/$8E/$1E/Turbo_ds);                    (* DS-Register von Turbo *)
  Move (Buffer, Screen, 4000);                      (* Bildschirm rest.      *)
  GotoXY (Old_X, Old_Y);                            (* Cursor setzen         *)
  Inline ($2E/$8E/$1E/Extern_ds/                    (* DS-Register rest.     *)
          $9D/$5D/$5F/$5E/$07/$1F/$5A/$59/$5B/$58/  (* Register rest.        *)
          $5D/                                      (* POP BP, POP BP        *)
          $8B/$E5/                                  (* MOV SP                *)
          $5D/                                      (* POP BP                *)
          $CF);                                     (* IRET                  *)
End;
```

Bild 3. Der INT-24H-Handler mit Sicherung und Restaurierung des Bildschirms

```
Procedure Int24_Status (New_State : Switch);

Type Registers = Record
       Case Boolean of
         True : (AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : Integer);
         false: (AL, AH, BL, BH, CL, CH, DL, DH : Byte)
       End;

Var Regs : Registers; (* Variable für Interrupt-Aufrufe *)

Begin
  If (New_State = On) and (State = Off) then (* Int24_Handler aktivieren *)
    Begin
      Regs.AH := $35;                 (* alten Int24-Vektor laden      *)
      Regs.AL := $24;                 (* Interrupt-Nummer              *)

      MSDOS (Regs);                   (* MS-DOS-Aufruf ausführen       *)
      Int24_Seg := Regs.ES;           (* Interrupt-Vektor sichern      *)
      Int24_Ofs := Regs.BX;
      Regs.AH := $25;                 (* neuen Int24-Vektor speichern *)
      Regs.AL := $24;                 (* Interrupt-Nummer              *)
      Regs.DS := CSeg;                (* Segment-Adresse               *)
      Regs.DX := Ofs (Int24);         (* Offset-Adresse                *)
      MSDOS (Regs);                   (* MS-DOS-Aufruf ausführen       *)
      Turbo_ds := DSeg;               (* Datensegment von Turbo        *)
      State := On                     (* neuer Zustand der Handlers    *)
    End;

  If (New_State = Off) and (State = On) then (* Int24_Handler desaktivieren *)
    Begin
      Regs.AH := $25;                 (* alten Int24-Vektor speichern *)
      Regs.AL := $24;                 (* Interrupt-Nummer              *)
      Regs.DS := Int24_Seg;           (* Segment-Adresse               *)
      Regs.DX := Int24_Ofs;           (* Offset-Adresse                *)
      MSDOS (Regs);                   (* MS-DOS-Aufruf ausführen       *)
      State := Off                    (* neuer Zustand der Handlers    *)
    End;
End;

(* -------------------------- Testprogramm -------------------------- *)

Var FID : File; (* Dummy-File-Identifier *)
    Key : Char; (* Dummy-Tastencode *)

Begin
  Int24_Status (On);  (* eigenen Int24_Handler aktivieren *)
  Assign (FID, 'A:DUMMYXYZ.ABC'); (* Datei ist hoffentlich nicht vorhanden *)
  ClrScr; (* Bildschirm löschen *)
  GotoXY (1,12); (* Cursor in Bildschirmmitte setzen *)
  WriteLn;
  WriteLn ('Bitte Klappe von Laufwerk A öffnen und beliebige Taste drücken');
  Read (Kbd, Key); (* Warten auf beliebige Taste *)
  WriteLn;
  {$I-} Reset (FID); {$I+} (* Datei zu öffnen versuchen *)
  If IOResult <> 0 then (* E/A-Fehler von Turbo erkannt ? *)
    WriteLn ('Gerät nicht bereit oder Datei nicht vorhanden !');
  Int24_Status (Off); (* eigenen Int24_Handler desaktivieren *)
End.
```

```
(* ------------------------ Int24 - Handler ------------------------- *)

Type Switch = (On, Off); (* mögliche Zustände des eigenen Int24_Handlers *)

Const Int24_Ofs : Integer = 0;      (* Offset des alten Int24-Handlers       *)
      Int24_Seg : Integer = 0;      (* Segment des alten Int24-Handlers      *)
                                    (* Reihenfolge der Konstanten wichtig !! *)
      State     : Switch  = Off;    (* aktueller Zustand des Int24_Handlers  *)

Procedure Int24;

Begin
  Inline ($B0/<3/                                (* MOV AL,3       *)
          $5D/                                   (* POP BP, POP BP *)
          $8B/$E5/                               (* MOV SP         *)
          $5D/                                   (* POP BP         *)
          $CF);                                  (* IRET           *)
End;

Procedure Int24_Status (New_State : Switch);

Type Registers = Record
       Case Boolean of
         True : (AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : Integer);
         false: (AL, AH, BL, BH, CL, CH, DL, DH : Byte)
       End;

Var Regs : Registers; (* Variable für Interrupt-Aufrufe *)

Begin
  If (New_State = On) and (State = Off) then (* Int24_Handler aktivieren *)
    Begin
      Regs.AH := $35;                (* alten Int24-Vektor laden      *)
      Regs.AL := $24;                (* Interrupt-Nummer              *)
      MSDOS (Regs);                  (* MS-DOS-Aufruf ausführen       *)
      Int24_Seg := Regs.ES;          (* Interrupt-Vektor sichern      *)
      Int24_Ofs := Regs.BX;
      Regs.AH := $25;                (* neuen Int24-Vektor speichern *)
      Regs.AL := $24;                (* Interrupt-Nummer              *)
      Regs.DS := CSeg;               (* Segment-Adresse               *)
      Regs.DX := Ofs (Int24);        (* Offset-Adresse                *)
      MSDOS (Regs);                  (* MS-DOS-Aufruf ausführen       *)
      State := On                    (* neuer Zustand der Handlers    *)
    End;

  If (New_State = Off) and (State = On) then (* Int24_Handler desaktivieren *)
    Begin
      Regs.AH := $25;                (* alten Int24-Vektor speichern *)
      Regs.AL := $24;                (* Interrupt-Nummer              *)
      Regs.DS := Int24_Seg;          (* Segment-Adresse               *)
      Regs.DX := Int24_Ofs;          (* Offset-Adresse                *)
      MSDOS (Regs);                  (* MS-DOS-Aufruf ausführen       *)
      State := Off                   (* neuer Zustand der Handlers    *)
    End;
End;
```

Bild 4. Dieses Programm bricht bei Ein-/Ausgabefehlern die Ein-/Ausgabeoperation ab

```
(* ------------------------- Testprogramm --------------------------- *)

Var FID : File; (* Dummy-File-Identifier *)
    Key : Char; (* Dummy-Tastencode *)

Begin
  Int24_Status (On);  (* eigenen Int24_Handler aktivieren *)
  Assign (FID, 'A:DUMMYXYZ.ABC'); (* Datei ist hoffentlich nicht vorhanden *)
  WriteLn;
  WriteLn ('Bitte Klappe von Laufwerk A öffnen und beliebige Taste drücken');
  Read (Kbd, Key); (* Warten auf beliebige Taste *)

  WriteLn;
  {$I-} Reset (FID); {$I+} (* Datei zu öffnen versuchen *)
  If IOResult <> 0 then (* E/A-Fehler von Turbo erkannt ? *)
    WriteLn ('Gerät nicht bereit oder Datei nicht vorhanden !');
  Int24_Status (Off); (* eigenen Int24_Handler desaktivieren *)
End.
```

```
(* ------------------------ Int24 - Handler ------------------------- *)

Type Switch = (On, Off); (* mögliche Zustände des eigenen Int24_Handlers *)
     Action = (Ignore, Retry, Terminate, Abort); (* Reaktionen des Handlers *)
                                                 (* Reihenfolge wichtig !!  *)

Const Int24_Ofs : Integer = 0;      (* Offset des alten Int24-Handlers       *)
      Int24_Seg : Integer = 0;      (* Segment des alten Int24-Handlers      *)
                                    (* Reihenfolge der Konstanten wichtig !! *)
      State     : Switch  = Off;    (* aktueller Zustand des Int24_Handlers  *)
      Disk      : Action  = Abort;  (* Antwort bei Disketten-Fehler          *)
      IO        : Action  = Abort;  (* Antwort bei allgemeinen E/A-Fehler    *)
      AX_Reg    : Integer = 0;      (* AX-Registerinhalt bei Int24-Aufruf    *)
      DI_Reg    : Integer = 0;      (* DI-Registerinhalt bei Int24-Aufruf    *)
      Response  : Byte    = 0;      (* Antwort des eigenen Int24_Handlers    *)
                                    (* Response = 0   --> ignorieren         *)
                                    (* Response = 1   --> wiederholen        *)
                                    (* Response = 2   --> Programm abbrechen *)
                                    (* Response = 3   --> Prozess abbrechen  *)

Procedure Int24;

Begin
  Inline ($50/$53/$51/$52/$1E/$06/$56/$57/$55/$9C/ (*     Register sichern *)
          $2E/$A3/AX_Reg/                           (* CS: MOV [AX_Reg],AX  *)
          $2E/$89/$3E/DI_Reg/                       (* CS: MOV [DI_Reg],DI  *)
          $FB);                                     (*     STI              *)

  If AX_Reg and $8000 = 0 then (* Disketten-Fehler ? *)
    Response := Ord (Disk) (* Antwort bei Disketten-Fehler *)
  else
    Response := Ord (IO);  (* Antwort bei allgem. E/A-Fehler *)
```

Bild 5. INT-24H-Handler unterscheidet Art des Fehlers

aufgerufen und zuletzt der Bildspeicher zurückkopiert. *Bild 3* zeigt die dazu nötigen Routinen, eingebunden in einem kleinem Testprogramm. Mit der Prozedur INT24H-Status kann die eigene Fehlerroutine (Prozedur INT24H) aktiviert bzw. desaktiviert werden. Zur Umlenkung des Interrupts 24H wird einfach die Adresse der INT-24H-Routine von MS-DOS zunächst gespeichert (u. a. für einen späteren Aufruf) und dann mit der Adresse der eigenen Prozedur überschrieben. Dadurch wird im Falle eines Ein-/Ausgabefehlers die Prozedur INT24H aufgerufen. Nach dem Sichern bzw. Laden diverser Register rettet die Routine den Bildspeicher und die Cursorposition, bevor die INT-24H-Routine von MS-DOS aufgerufen wird. Nach diesem Aufruf gilt es den Bildspeicher zurückzukopieren und den Cursor an seine alte Position zu bringen. Nach dem Restaurieren bzw. Umladen einiger Register endet die Routine mit einem IRET-Befehl (Return from Interrupt). Es sei an dieser Stelle darauf hingewiesen, daß sich die Routine in der vorliegenden Form nur zum Sichern des Bildes einer Monochrom-Karte eignet. Bei einem Ein-/Ausgabefehler im Grafikmodus versagt diese Routine ebenfalls. In diesem Falle muß ein anderer Weg beschritten werden.

```
  Inline ($9D/$5D/$5F/$5E/$07/$1F/$5A/$59/$5B/$58/ (*     Register rest.   *)
         $8B/$E5/                                  (*     MOV SP,BP        *)
         $5D/                                      (*     POP BP           *)
         $2E/$A0/Response/                         (* CS: MOV AL,[Resonse] *)
         $CF);                                     (*     IRET             *)
End;

Procedure Int24_Status (New_State : Switch; Disk_Error, IO_Error : Action);

Type Registers = Record
       Case Boolean of
         True : (AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : Integer);
         false: (AL, AH, BL, BH, CL, CH, DL, DH : Byte)
       End;

Var Regs : Registers; (* Variable für Interrupt-Aufrufe *)

Begin
  If (New_State = On) and (State = Off) then (* Int24_Handler aktivieren *)
    Begin
      Regs.AH := $35;           (* alten Int24-Vektor laden     *)
      Regs.AL := $24;           (* Interrupt-Nummer             *)
      MSDOS (Regs);             (* MS-DOS-Aufruf ausführen      *)
      Int24_Seg := Regs.ES;     (* Interrupt-Vektor sichern     *)
      Int24_Ofs := Regs.BX;
      Regs.AH := $25;           (* neuen Int24-Vektor speichern *)
      Regs.AL := $24;           (* Interrupt-Nummer             *)
      Regs.DS := CSeg;          (* Segment-Adresse              *)
      Regs.DX := Ofs (Int24);   (* Offset-Adresse               *)
      MSDOS (Regs);             (* MS-DOS-Aufruf ausführen      *)
      Disk := Disk_Error;       (* Antwort bei Disketten-Fehler *)
      IO := IO_Error;           (* Antwort bei allgem. Fehler   *)
      State := On               (* neuer Zustand der Handlers   *)
    End;

  If (New_State = Off) and (State = On) then (* Int24_Handler desaktivieren *)
    Begin
      Regs.AH := $25;           (* alten Int24-Vektor speichern *)
      Regs.AL := $24;           (* Interrupt-Nummer             *)
      Regs.DS := Int24_Seg;     (* Segment-Adresse              *)
      Regs.DX := Int24_Ofs;     (* Offset-Adresse               *)
      MSDOS (Regs);             (* MS-DOS-Aufruf ausführen      *)
      State := Off              (* neuer Zustand der Handlers   *)
    End;
End;

(* --------------------------- Testprogramm --------------------------- *)

Var FID : File; (* Dummy-File-Identifier *)
    Key : Char; (* Dummy-Tastencode *)

Begin
  Int24_Status (On, Abort, Retry);  (* eigenen Int24_Handler aktivieren *)
  Assign (FID, 'A:DUMMYXYZ.ABC'); (* Datei ist hoffentlich nicht vorhanden *)
  WriteLn;
  WriteLn ('Bitte Klappe von Laufwerk A öffnen und beliebige Taste drücken');
  Read (Kbd, Key); (* Warten auf beliebige Taste *)
  WriteLn;
  {$I-} Reset (FID); {$I+} (* Datei zu öffnen versuchen *)
  If IOResult <> 0 then (* E/A-Fehler von Turbo erkannt ? *)
    WriteLn ('Gerät nicht bereit oder Datei nicht vorhanden !');
  Int24_Status (Off, Abort, Retry); (* eigenen Int24_Handler desaktivieren *)
End.
```

Eigentlich ist es gar nicht nötig, innerhalb der INT-24H-Routine eine Fehlermeldung auszugeben und eine Abfrage durchzuführen. *Bild 4* zeigt ein Turbo-Programm mit einer INT-24H-Prozedur, die auf jeden Ein-/Ausgabefehler mit dem Wert 3 im AL-Register reagiert. Diese Antwort führt nur zu einem Abbruch der betreffenden Ein-/Ausgabeoperation. Bei einem so abgebrochenen Diskettenzugriff zeigt die Turbo-Funktion IOResult einen Fehler an. Die gleiche Funktion erkennt jedoch keinen Fehler bei einer abgebrochenen Ein-/Ausgabeoperation auf ein Peripheriegerät. So würden also im Falle eines nicht bereiten Druckers der zu druckende Text ohne Feldermeldung verloren gehen. Bei Programmen mit Zugriff auf Peripheriegeräte muß die INT-24H-Routine je nach Zugriffsart individuell reagieren.

Das Programm in *Bild 5* erlaubt bei der Aktivierung des eigenen INT-24H-Handlers die Antwort auf einen Fehler bei einem Diskettenzugriff oder bei einem allgemeinen Ein-/Ausgabefehler individuell zu bestimmen. Es hat sich als günstig erwiesen, fehlgeschlagene Diskettenzugriffe abzubrechen, während man bei allgemeinen Ein-/Ausgabefehlern die Operation wiederholt. Bei der Ausgabe auf einen nicht bereiten Drucker wiederholt das Programm die Ausgabe so lange, bis der Drucker betriebsbereit ist. Dadurch kann u. U. der Eindruck entstehen, daß das Programm sich „aufgehängt" hat. In einer solchen Situation sollte man die benötigten Peripheriegeräte auf Betriebsbereitschaft hin untersuchen.

Zum Schluß gilt es noch einige Worte über die drei Beispiele zu verlieren. Die Umlenkung des Interrupts 24H ist eine relativ komplizierte Angelegenheit und erfordert gute Programmierkenntnisse.

Selbst die kleinste Änderung an der Prozedur INT24H oder am Deklarationsteil kann schwerwiegende Folgen nach sich führen. Als unerfahrener Programmierer sollten Sie am besten eine der drei Routinen unverändert übernehmen.

Ulrich Cebulla

Grafik mit EGA

Teil 4: Diesmal auch mit Text

Die Forderung, im Grafikmodus den Text an jede beliebige x/y-Position auf dem Bildschirm stellen zu können, sollte von jeder guten Grafikkarte erfüllt werden. Ebenso sollte der Textform, z. B. Größe, Abstand und Richtung, ein möglichst großer Spielraum zur Verfügung stehen. Allgemein kann wohl gesagt werden, daß im Grafikmodus an den Text innerhalb einer Grafik höhere Anforderungen gestellt werden als im normalen Textmodus. Wie sich hier nun die IBM-EGA verhält, und welche Möglichkeiten sie bietet, soll in diesem Artikel etwas näher untersucht werden.

Schaut man sich die Funktion einer Grafikkarte im Textmodus und im Grafikmodus etwas näher an, so stellt man fest, daß sich die beiden Arbeitsweisen sehr stark unterscheiden. Das läßt den Schluß zu, daß die Kombination von Textmodus und Grafik oder Grafikmodus und Text einige Probleme aufwerfen muß. Hierfür sind die Bilder der Sinus-Funktion im letzten Heft ein guter Beweis. Für den Text in der Grafik liegt die Zeilenzahl mit maximal 25 Zeilen bei der Verwendung von „Gotoxy" und „Write/ln" ziemlich fest. Weiterhin erscheint der Text, abhängig von der verwendeten Farbe, nicht nur in einer Speicherebene, es sei denn, man verwendet nur die Grundfarben blau (C0), grün (C1), rot (C2) oder grau (C3) (bei der Standard-Initialisierung des Attribute Controllers). Selbst wenn man nur die Grundfarben verwendet ist das für die Grafik von Nachteil, da jetzt im Hintergrund des Textes Grafik nur noch schwierig unterzubringen ist. Vorher jedoch noch ein paar kurze Erklärungen zur Speicherorganisation und -steuerung der IBM-EGA, die in *Bild 1 und 2* grafisch dargestellt sind. Bild 1 zeigt die logische Verknüpfung der wichtigsten Grafik Controller Register mit dem Bildschirmspeicher z. B. bei einer Schreiboperation. Folgende Register sind dabei zu beachten:

Set/Reset Register,
Enable Set/Reset Register,
Bit Mask Register und die
32 Bit Prozessor LTs.

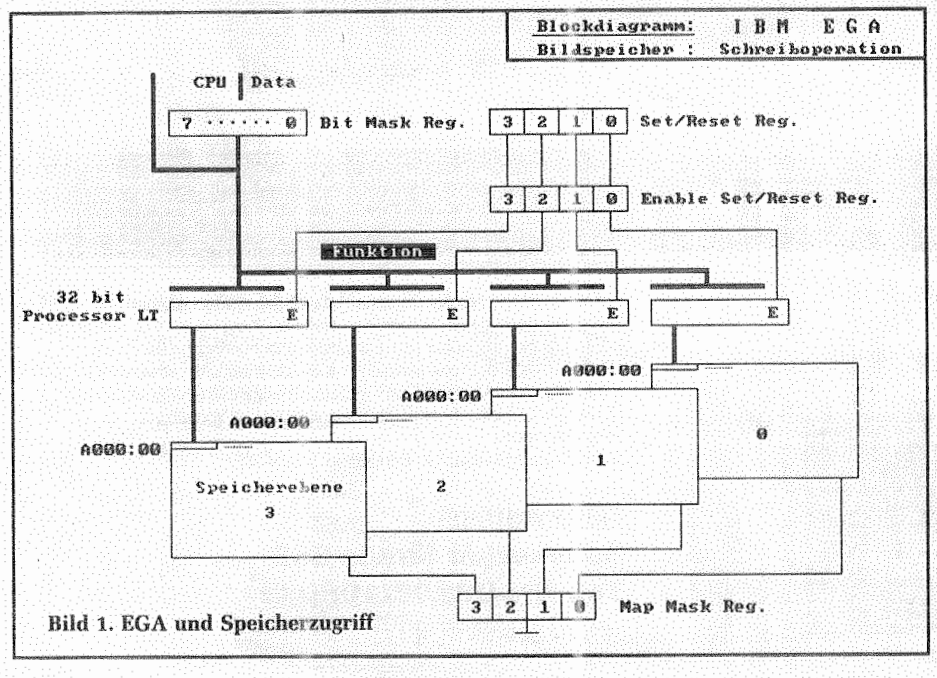


Bild 1. EGA und Speicherzugriff

Die Speicherebenen 0...3 und das Map Mask Register des Sequencers sind betroffen (siehe auch Beschreibung der Grafik und Attribute Controller Register in mc, Ausgabe 2/87).
Der Übersichtlichkeit wegen sind die Funktionen AND, OR, XOR nur angedeutet. Das Bild 1 zeigt deutlich, daß bei allen Schreiboperationen nur in die Speicherebenen geschrieben werden kann, deren zugehöriges Bit im Map Mask Register gesetzt ist. Weiterhin soll das „E" in den 4×8-Bit-Registern (32 Bit Prozessor LTs) deutlich machen, daß im „Write Mode 0" nur in die Speicherebenen das entsprechende Bit aus dem Set/Reset Register geschrieben wird, die auch durch das Enable Set/Reset Register erlaubt sind. Alle nicht erlaubten Speicherebenen erhalten ihre Daten von der CPU. Grundsätzlich entscheidet dann noch das Bit Mask Register, ob die Daten aus den Processor LTs und somit vom letzten Memory Read stammen oder nicht. Das Bild 2 zeigt, wie die Palette Register des Attribute Controllers adressiert werden. Ist zum Beispiel nur das Bit C0 im Color Plane Enable Register des Attribute Controllers gesetzt, so werden auch nur das Palette Register 1 oder 0 adressiert, das heißt, nur die Speicherebene 0 ist auf dem Bildschirm sichtbar. Sind beispielsweise das Bit C0 und C2 gesetzt, also Speicherebene 0 und 2 auf dem Bildschirm sichtbar, so werden jetzt die Palette Register 0,1,4 und 5 adressiert. Normalerweise sind immer alle 4 Bits in C0 bis C3 gesetzt, so daß alle 16 Palette Register adressiert werden können, d. h., alle 4 Speicherebenen (16 Farben) sind auf dem Bildschirm sichtbar. Bei den EGAs mit 64 KByte Speicher sind im EGA Modus 0F (Monochrome) nur C0 und C2 gültig, da ja auch nur die Speicherebenen 0 und 2 zur Verfügung stehen. Jetzt werden allerdings die Palette Register 0,1,4,5 durch die geraden (even) CPU-Adressen, die Palette Register 8,9,C,D durch die ungeraden (odd) CPU-Adressen erreicht. Weshalb die Register 8,9,C und D auch noch adressiert werden, soll an dieser Stelle nicht weiter erläutert werden. Nur soviel sei gesagt: Setzt man ins Register C den gleiche Wert wie in Register 4 (siehe Attribute Register in Heft 2/87), so schaltet man von Blinking nach Intensified Video um.

Grafikmodus und Text

Die Routine „Write(ln)" in Turbo Pascal benutzt grundsätzlich das BIOS. Im Grafikmodus wird vom BIOS das Map Mask Register des Sequencers für die Farbe des Textes verwendet. Das bedeutet, daß der Text immer in alle Speicherebenen geschrieben wird. Will man das unter bestimmten Umständen vermeiden, so muß man seine eigene Routine „Gotoxy, Write(ln)" schreiben. Da das Turbo Pascal hierfür eine besondere Methode bereithält, ist das sehr einfach (siehe Turbo-Pascal-Handbuch, „User Written I/O Drivers"). Die Variable „ConOutPtr" kann dazu wie folgt benutzt werden:

ConOutPtr := ofs(Writetext);

„Writetext" ist hier der Name der Prozedur, die nun den Text an die entsprechende xy-Position in den EGA Speicher schreiben soll. *Bild 3* zeigt das Programm dazu. Diese Prozedur soll nur zeigen, wie der Text im Grafikmodus in den Bildschirmspeicher geschrieben werden kann, und zwar auch hier in alle Bildspeicherebenen gleichzeitig. Die erste Instruktion in der „for loop"

mem(egabase:offset):=0;

sorgt dafür, daß alle Speicherebenen dieser Byteposition gelöscht werden. In der folgenden Instruktion wird dann die gewählte Farbe in das „Map Mask Register" des Sequencers gesetzt. Anders gesagt, mit der Farbe wird/werden die Speicherebene/n ausgewählt, in die mit der nachfolgenden Memory-Instruktion dann das Font Pattern in die Byteposition geschrieben wird. Die nächste Instruktion

```
procedure writechar(c :byte);   { Schreibe den Charakter "c" an die Position  }
                                { x/y (x=xtext, y=ytext). Die var "pattern"   }
                                { gibt an, aus wievielen Font-Bytes ein Cha-  }
                                { rakter besteht, z.B. 8 für den 8x8 Font.    }
                                { x=1..80, y=1..35, und box=10. egabase=$A000,}
                                { fontsegm:fontoffs=Fontadresse im EGA ROM     }
var offset,i : integer;
   begin
   offset := xtext - 1 + ((ytext - 1) * 80*box); { offset in den Bildspeicher }

   port[$3c4]:=2;               { Map Mask Register des Sequencers }

   for i := 0 to pattern - 1
   do begin
      mem[egabase:offset]:=0;   { alle Speicherebenen dieser Position löschen }
      port[$3c5]:=farbe;        { Farbe des Charakters ins Map Mask Register  }
                                { Fontpattern schreiben                       }
      mem[egabase:offset]:=mem[fontsegm:fontoffs + (c*pattern) + i];

      port[$3c5]:=$0F;          { erlaube wieder alle Speicherebenen          }
      offset:= offset + 80;     { nächste scan line                           }
   end;

end;
```

Bild 3. Das Programm, das den Text an die x-y-Position bringt

port($3C5):=$0F;

erlaubt wieder alle Speicherebenen, so daß mit der ersten Instruktion in der „for loop" wiederum alle Speicherebenen gelöscht werden können. Die Instruktion

offset:=offset + 80;

weist jetzt auf die folgende Scan Line, in die das nächste Font Pattern geschrieben werden soll. Die Laufvariable „i" ist die Byteadresse der einzelnen Fontbytes eines Zeichens.

Horizontal liegt die Position der einzelnen Zeichen immer auf der Bytegrenze im Bildschirmspeicher. Ein Grund dafür ist, daß auch die Fonts (8×8 und 8×14) Byte-orientiert sind. Vertikal in der y-Achse kann ein Zeichen in jeder beliebigen Scan Line beginnen. Das bedeutet, daß bei der Gotoxy-Instruktion x-Werte von 1 bis 80, und y-Werte von 1 bis 350 (oder auch darüber) im Grafikmodus möglich sind. Da „write(ln)" aber nicht nur Text überträgt sondern auch Kontrollzeichen wie zum Beispiel BS (Back Space, Rücksetzen), LF (Line Feed, Zeilentransport) und CR (Carriage Return, Wagenrücklauf), muß die Prozedur „writechar" diese Funktionen auch unterstützen. Die Gotoxy-Instruktion ist dann nur noch eine Formsache. Wie nun Text und Grafik im Grafikmodus zusammenspielen können, soll das folgende Programmbeispiel zeigen.

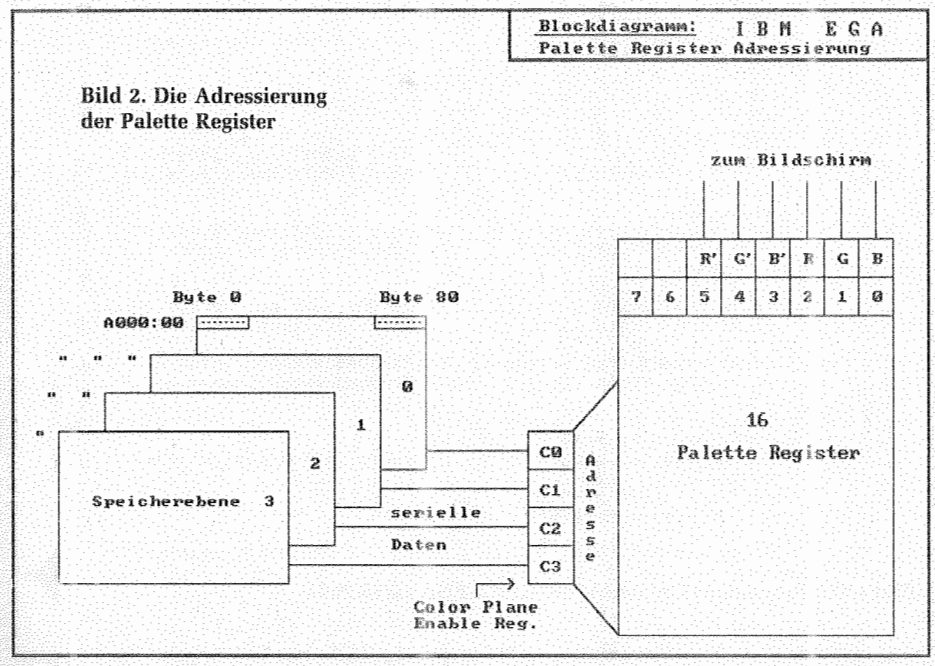


Bild 2. Die Adressierung der Palette Register

Programm „EGA-DRAW"

Das Programm EGA-DRAW (oder deutsch: „Zeichnen mit der IBM-EGA", *Bild 4*) will versuchen, einige der vielfältigen Möglichkeiten der IBM-EGA im Grafikmodus zu demonstrieren. Alle Bilder in diesem Artikel sind mit diesem Programm hergestellt worden. Das Beispielprogramm benutzt den EGA-Modus 0F/10, also Grafik 640×350 mit dem Monochrom oder Enhanced-Color-Bildschirm. Das Programm erlaubt das Zeichnen auf dem Bildschirm mit Hilfe der vier Cursor-Tasten. Der Bildspeicher hat eine Größe von 640 × 406 Punkten. Es werden nur zwei der vier möglichen Speicherebenen benutzt, so daß das Programm auch auf einer IBM-EGA mit nur 64 KByte laufen kann. In der Speicherebene C2 wird ein kleines Fadenkreuz

bewegt, während am Bildrand die Komunikation mit dem Programm abläuft (z. B. Texteingabe, Zeichnen, Radieren, Scrollen usw.). In der Speicherebene C0 wird das Bild erzeugt. *Bild 5* zeigt die Tastenbelegung und die Funktionen, die hier noch erläutert sind:

h

Wird das Program EGA-DRAW ausgeführt, so erscheint als erstes das Bild 5 auf dem Bildschirm. Es zeigt die Tastenbelegung und deren Funktionen. Mit der Taste „h" kann nun zwischen diesem Bild und der Zeichenebene hin- und hergeschaltet werden. Das Bild 5 wird immer von der Diskette geladen.

E

Programm EGA-DRAW beenden.

c

Bildschirm mit den beiden Cursor-Tasten Up/Down Scan Line für Scan Line nach oben oder unten steuern. Nach 56 scans ist die Scan Line 406 am unteren Bildrand und weiteres Scrollen wird unterdrückt.

l

Laden eines auf der Diskette gespeicherten Bildes in den Bildschirmspeicher. In Zeile 1 muß der volle Dateiname eingegeben werden.

s

Speichern des Bildes in der Zeichenebene auf der Diskette. In Zeile 1 muß der Dateiname eingegeben werden.

m

Bild horizontal/vertikal mit den vier Cursor Tasten rotieren. Die Fadenkreuzebene bleibt dabei stehen.

p

Bild der Zeichenebene mit der Prozedur „prtscr" auf dem Matrixdrucker ausdrucken.

d

Zeichnen. Am unteren Bildrand erscheint „zeichnen" oder „löschen". Wird nun das Fadenkreuz bewegt, so wird in der Zeichenebene entweder gezeichnet oder gelöscht. Die Anzahl der Punkte, die mit einem Tastendruck gezeichnet/gelöscht werden (Geschwindigkeit) wird am unteren Bildrand angezeigt. Mit der Taste d kann das „Zeichnen" ein- oder ausgeschaltet werden. (Fadenkreuz bewegen ohne zu zeichnen).

r

Umschalten zwischen „Zeichnen" und „Löschen".

cursor

Bewegen des Fadenkreuzes in alle Richtungen.

1..9,0

Anzahl der Punkte, die mit einem Cursor Tastendruck gezeichnet werden.

(0=10 Punkte).

```
Program EGAdraw;
{$I grafik.ega}              { IBM EGA Unterstützungs-Programme }
{$I prtscr.ega}              { Bildschirm drucken }
{$I saveload.ega}            { Bildspeicher lesen/schreiben, Diskette }
var
   x,y,x1,y1:integer;        { x/y Koordinaten }
   savestart:integer;        { Startzeile }
   sr:byte;                  { schreiben/radieren }
   draw,savedraw,xy,help,fontsw,text,invertext,vertext:boolean;
   speed,savespeed:byte;     { Fadenkreuz Geschwindigkeit }
   fdkz,bld1:byte;           { Fadenkreuz-/Zeichnung-Speicherebene }
   bildfd,bild:byte;         { Fadenkreuz-/Zeichnung-Bildebene }
   readebene:byte;           { Auswahl der zu lesenden Bildspeicherebene. }
   linebuff:integer;         { Freie Bildschirmzeile }
   taste,texttaste:char;
{-------------------------------------------------------------------------------}
Procedure display;
begin
  gotoxy(47,35); write(speed,' Punkte  '); { Fadenkreuz-Geschwindigkeit       }
end;
{-------------------------------------------------------------------------------}
Procedure fadenkreuz(lr,ud:integer);    { Fadenkreuz Bewegung, Bild zeichnen  }
var speedloop:integer;
begin
     if not help
     then begin
          if not draw
          then
              speedloop:=1                    { schnelle loop }
          else
              speedloop:=speed;               { pel für pel loop }

          For i:= 1 to speedloop
          do begin
             {==================zeichnen=====================}
             if draw
             then begin
                  speicherebene(bld1);        { Speicherebene der Zeichnung }
                  pset(x1,y1,sr,0);           { Bild zeichnen  }
                  speicherebene(fdkz);        { Speicherebene des Fadenkreuzes }
                  end;
             {===============================================}
             y:=y1;
             for x:=x1-3 to x1+3
                 do pset(x,y,0,0);            { altes Fadenkreuz löschen }
             x:=x1;
             for y:=y1-3 to y1+3              {   "         "       "    }
                 do pset(x,y,0,0);
             if draw
             then begin
                  x1:=x1+lr;                  { neue x-Position, langsam }
                  y1:=y1+ud;                  { neue y-Position, langsam }
                  end
             else begin
                  x1:=x1+(lr*speed);          { neue x-Position, schnell }
                  y1:=y1+(ud*speed);          { neue y-Position, schnell }
                  end;

             if x1 > 639 then x1:=639;        { Randbegrenzung, rechts }
             if x1 < 0   then x1:=0;          { Randbegrenzung, links }

             if y1 > vert then y1:=vert;      { Randbegrenzung, unten }
             if y1 < 0   then y1:=0;          { Randbegrenzung, oben }

             y:=y1;
             for x:=x1-3 to x1+3
                 do pset(x,y,15,0);           { neues Fadenkreuz setzen }
             x:=x1;
             for y:=y1-3 to y1+3
                 do pset(x,y,15,0);           {   "         "       "    }

             if xy and not text                    { x/y Werte anzeigen }
```

Bild 4. Das Programm EGA-DRAW mit Grafik EGA als Include

```
            then begin
                 gotoxy(1,20); Write(xi,'  ');   ( x-Wert horizontal anzeigen)
                 wrthor:=false;
                 gotoxy(1,22); Write(y1,'  ');   ( y-Wert vertikal anzeigen)
                 wrthor:=true;
                 end;
            end;
       end;
end;
(-------------------------------------------------------------------------------)
Procedure textproc;                          ( Bildtext Procedure )
var xt,yt:integer;
begin
     if not help
     then begin
          savedraw:=draw;                    ( Zeichnen-Modus festhalten )
          draw:=false;                       (                           )
          savespeed:=speed;                  (                           )

          speed:=1;
          text:=true;                        ( Textmodus ein )

          if fontsw                          ( Text-Font Auswahl )
          then
              fontpointer(false);            ( 8x14 font )

          gotoxy(32,1);
          write(' Textmodus');               ( Modus in Zeile 1 setzen )

          if fontsw                          ( 8x14 / 8x8 Font )
          then
              write(' 8x14')
          else
              write(' 8x8');

          if invertext                       ( invertiert schreiben )
          then begin
               wrtinvert:=true;              ( siehe Procedure  "writechar" )
               write(' invert ');
               end;

          if vertext                         ( vertikal schreiben )
          then begin
               write(' vertikal');
               wrthor:=false;                ( siehe Procedure "writetext" )
               end;

          graftext:=true;                    ( erlaube y-Achse von 0 .. 405 )
                                           ( siehe Procedure "writechar/writetext")

          xt:=trunc(x1/8)+1;                 ( xt auf Bytegrenze setzen )
          fadenkreuz(-(x1-(xt-1)*8),0);      ( Fadenkreuz auf Bytegrenze setzen )
          if y1 > (405-box)
          then begin
               yt:=vert-box+1;
               fadenkreuz(0,-(y1-yt));       ( Fadenkreuz y-max setzen )
               end
          else
              yt:=y1;                        ( y-Ache max vert - box )

          gotoxy(xt,yt);                     ( 1. x/y-Position des Textes setzen )

          (=================Text-Loop=========================)

          repeat
               read(kbd,texttaste);          ( Taste lesen          )
               speicherebene(bld1);          ( Speicherebene Zeichnung)
               write(texttaste);             ( Charakter schreiben )
               speicherebene(fdkz);          ( Speicherebene Fadenkreuz )
                          ( Fadenkreuz im Textmodus )
           if nextxy and not (texttaste = chr(13))
                then begin
                     if texttaste = chr(8)       ( BS ? (Backspace))
                     then if wrthor
                          then
                              fadenkreuz(-8,0)   ( BS 1 Byte  <= )
                          else
                              fadenkreuz(0,-box) ( BS 1 Char box nach oben )
                     else if wrthor
                          then
                              fadenkreuz(8,0)    ( nächster Charakter => )
                          else
                              fadenkreuz(0,box); ( nächster Char nach unten )
                     end;
          until texttaste = chr(13);  ( CR = Text-Ende      )
          (=================================================)

          graftext:=false;                ( zurück zum Zeichnen-Modus )
          wrtinvert:=false;
          wrthor:=true;
          draw:=savedraw;

          gotoxy(32,1);
          write('                                ');  ( Modus löschen )

          if fontsw
          then
              fontpointer(true);          ( zurück nach 8x8 font )
          fadenkreuz(0,0);
          speed:=savespeed;
          display;
          text:=false;                    ( Textmodus aus )
          end;
end;
(-------------------------------------------------------------------------------)
Procedure drawproc;                       ( zeichnen / nicht zeichnen )
begin
if not help
then begin
     if draw

     then begin
          draw:= not draw;
          gotoxy(38,35);
          write('          ');
          end
     else begin
          draw:= not draw;                ( Wenn "draw" hier true, dann kann ge- )
          gotoxy(38,35);                  ( zeichnet werden.                     )
          if sr=15                        ( Wenn "sr" (set/reset) = 15 (Farbe),  )
          then                            ( dann wird gezeichnet, sonst glöscht  )
              write('zeichnen ')
          else
              write(' löschen ');
          end;

     write(speed,' Punkte  ');
     end;
end;
(-------------------------------------------------------------------------------)
Procedure setrstproc;                     ( Punkt setzen oder löschen )
begin                                     ( Punkt setzen: "sr" = 15   )
     if sr = 0                            ( Punkt löschen:"sr" = 0    )
     then
         sr:=15                           ( sr (set/reset) Farbe für bildzeichnen )
     else                                 ( in der Procedure "fadenkreuz".        )
          sr:=0;
     if draw
     then begin
          gotoxy(38,35);
          if sr=15
```

```
          then
              write('zeichnen ')
          else
              write(' löschen ');
          end;
     gotoxy(47,35); write(speed,' Punkte  ');
end;
{-------------------------------------------------------------------------}
Procedure diskbild(lesen:boolean);   { Bildschirm lesen/schreiben, Diskette
type name = string[12];
var diskname:name;
    move    :byte;
begin
     gotoxy(1,2);
     write('                         ');     { IO Error Zeile löschen }
     if lesen
     then begin
          gotoxy(32,1);
          write(' Diskette lesen');
          end
     else begin
          gotoxy(32,1);
          write(' Diskette schreiben');
          end;
     gotoxy(1,1);
     if lesen
     then
          write('L_Name:              ')
     else
          write('S_Name:              ');
     gotoxy(9,1);
     readln(diskname);                    { Filename für Lesen/Schreiben des}
     if lesen                             { Bildschirmspeichers, (Diskette) }
     then begin
          speicherebene(bld1);            { Bildspeicherebene der Zeichnung }
          loadscreen(diskname);           { Bild laden von der Diskette }
          speicherebene(fdkz);            { zurück zu Fadenkreuz-Speicherebene }
          end
     else begin                           { erlaube nur lesen der Bildspeicher-}
          speicherebeneread(readebene);   { ebene in der die Zeichnung steht.  }
          savescreen(diskname);           { Bild auf Diskette speichern }
          end;
     gotoxy(32,1);
     write('                    ');

end;
{-------------------------------------------------------------------------}
Procedure scrollproc;              { Bildschirm scrollen. }
begin
     gotoxy(32,1);
     write(' Scrollmodus ');
     scroll(1,false);              { eine scan line scrollen }
     gotoxy(32,1);
     write('             ');
end;
{-------------------------------------------------------------------------}
Procedure movebild;                { Bild horizontal/vertikal im Bildspeicher  }
var i,j :integer;                  { bewegen (rotieren).                       }
begin
     gotoxy(32,1);
     write(' Movemodus ');
     if EGA64                 { j = Adressindex (horizontal min 1 Byte)       }
     then
          j := 2              { bei EGA64 lassen sich nur even Bytes nach even}
     else                     { Bytes und odd Bytes nach odd bytes über die   }
          j := 1;             { 32 bit Processor LT's im EGA-Speicher übertragen.}

EGAwritemode(1);               { 32 bit read/write Modus (write mode 1) }
speicherebene(bld1);
linebuff:= (vert+2)*80;        { nicht benutzter Speicher im Bildschirmspeicher }
                               { wird als buffer genutzt.                       }
```

```
{=====================Loop Bild horizontal/vertikal rotieren=================}
repeat
     read(kbd,taste);
      case ord(taste) of
       75:   for i:=0 to vert          { Bild horizontal 1/(2) Bytes nach links }
             do begin
                move(mem[egabase:(i*80)],mem[egabase:linebuff],j);
                move(mem[egabase:(i*80+j)],mem[egabase:(i*80)],(80-j));
                move(mem[egabase:linebuff],mem[egabase:(i*80+(80-j))],j);
                end;
       77:   for i:=0 to vert          { Bild horizonzal 1/(2) Bytes nach rechts}
             do begin
                move(mem[egabase:(i*80+(80-j))],mem[egabase:linebuff],j);
                move(mem[egabase:(i*80)],mem[egabase:(i*80+j)],(80-j));
                move(mem[egabase:linebuff],mem[egabase:(i*80)],j);
                end;
       72: begin                       { Bild vertikal eine scan line nach oben }
             move(mem[egabase:000],mem[egabase:linebuff],80);
             move(mem[egabase:080],mem[egabase:000],32400);
             move(mem[egabase:linebuff],mem[egabase:(vert*80)],80);
             end;
       80: begin                       { Bild vertikal eine scan line nach unten}
             move(mem[egabase:(vert*80)],mem[egabase:linebuff],80);
             move(mem[egabase:000],mem[egabase:080],32400);
             move(mem[egabase:linebuff],mem[egabase:000],80);
             end;
       end;
until taste = 'm';                 { Loop mit Taste "m" verlassen }
speicherebene(fdkz);
EGAwritemode(0);                   { zurück zu write mode 0 }

     gotoxy(32,1);
     write('             ');
end;

{-------------------------------------------------------------------------}
Procedure printscr;                  { Bildspeicherebene drucken }
begin
     gotoxy(32,1);
     write(' Bildschirm drucken ');
     prtscr(1,vert+1);               { Speicherebene C0 drucken }
     gotoxy(32,1);
     write('                    ');
end;

{-------------------------------------------------------------------------}
Procedure helpproc;                  { Tastenbelegung (HELP) auf dem Bildschirm  }
                                     { anzeigen.                                 }
begin                                { Ist die Help-Bildseite nicht vorhanden,   }
     help:=not help;                 { sollte verhindert werden,daß diese Procedur }
     if help                         { ausgeführt wird.                          }
     then begin
          Bildebene(bildfd);           { nur Bildebene des Fadenkreuzes anzeigen  }
          savestart:=startzeile;       { alte Startzeile (sroll) speichern        }
          startzeile:=0;
          scroll(0,true);              { Startzeile 0000 in "scroll" Proc. setzen }
          loadscreen('help.scr');      { Help-Bildseite laden (wenn vorhanden)    }
          end
     else begin
          startzeile:=savestart;
          scroll(0,true);              { alte Startzeile in "scroll" Proc. laden }
          Bildebene(bild);             { nur Bildebene der Zeichnung anzeigen    }

          fillchar(mem[$A000:00],32479,0); { Fadenkreuz-Speicherebene löschen}
                                           { 406 scans * 80 Bytes }
          draw:=true;
          drawproc;
          fadenkreuz(0,0);
          end;
end;
{-------------------------------------------------------------}
```

```
{                                Programm Start                                }
{-----------------------------var Initialisierung-----------------------------}

begin
startzeile:=0;                  { Startzeile des Bildschirmes }
x1:=320;                        { Startposition des Fadenkreuzes }
y1:=175;                        { in der Bildschirmmitte        }
speed :=10;                     { cursor Geschwindigkeit 10 Punkte/ Tastedruck)
draw:=false;
xy:=false;
sr:=15;                         { Pset/Prest Farbe }
help:=false;
fontsw:=false;
invertext:=false;
vertext:=false;

{------------------------------------------------------------------------------}
graphics(true);                 { Start Grafik }
{------------------EGA/Bildschirm Auswahl--------------------------------------}
if EGAmode = $0F                { Monochrom Display und 64 - 256k Memory }
then begin
     farbpalette($18,12); { Blinking ausschalten }
{ soll das Blinken bei Überlappung ausgeschaltet werden, }
     farbpalette($08,13); (Punktüberlappung: Blinking intensified/normal video)
{ so muß die Instruktion verhindert werden }

     fdkz:=12;                  { Fadenkreuz/Modus-Information in Speicherebene C2 }
     bld:=3;                    { Zeichnung: Speicherebene C0 }
     bildfd:=4;                 { Bildebene: Fadenkreuz }
     bild:=5;                   { Bildebene: Zeichnung und Fadenkreuz)
     readebene:=0;              { nur Bildspeicherebene C0 lesen.)
     end
else if EGA64                   { Enhanced Color Display und 64k Memory }
     then begin
          farbpalette($35,4);  { Fadenkreuz/Modus : Farbe pink (even))
          farbpalette($35,12); { Fadenkreuz/Modus : Farbe pink (odd))
          farbpalette($24,1);  { Zeichnung        : Farbe rot (even))
          farbpalette($24,9);  { Zeichnung        : Farbe rot (odd))
          farbpalette($3e,5);  { Punktueberlappung: Farbe gelb (even))
          farbpalette($3e,13); { Punktueberlappung: Farbe gelb (odd))
          fdkz:=12;         { Fadenkreuz/Modus-Information in Speicherebene C2 }
          bld:=3;           { Zeichnung: Speicherebene C0 }
          bildfd:=4;        { Bildebene: Fadenkreuz }
          bild:=5;          { Bildebene: Zeichnung und Fadenkreuz }
          readebene:=0;     { nur Bildspeicherebene C0 lesen.)
          end
     else begin                  { Enhanced Color Display und Memory > 64k }
          farbpalette($35,4); { Fadenkreuz/Modus : Farbe pink }
          farbpalette($24,1); { Zeichnung        : Farbe rot }
          farbpalette($3e,5); { Punktueberlappung: Farbe gelb }
          fdkz:=4;          { Fadenkreuz/Modus-Information in Speicherebene C2 }
          bld:=1;           { Zeichnung: Speicherebene C0 }
          bildfd:=4;        { Bildebene: Fadenkreuz }
          bild:=5;          { Bildebene: Zeichnung und Fadenkreuz)
          readebene:=0;     { nur Bildspeicherebene C0 lesen.)
          end;
{------------------------------------------------------------------------------}

vert:=405;                      { Bildhöhe 406 scans }
graftext:=false;                { normaler Text }
speicherebene(fdkz);            { Speicherebene, Fadenkreuz }
fadenkreuz(0,0);                { 1.mal Fadenkreuz setzen }
drawproc;                       { zeichnen einschalten }
helpproc;                       { HELP.SCR file anzeigen, (diese Instruktion nur aus-}
                                { führen, wenn die File vorhanden ist. }

{====================Auswahl-Loop der Tastenfunktionen=======================}
repeat
     read(kbd,taste);

       case ord(taste) of
       75: fadenkreuz(-1,0);                { nach links                       }
       77: fadenkreuz(1,0);                 { nach rechts                      }
       72: fadenkreuz(0,-1);                { nach oben                        }
       80: fadenkreuz(0,1);                 { nach unten                       }
      104: helpproc;                        { h = help Bild                    }
      105: invertext:=not invertext;        { i = invertiert text schreiben    }
      112: printscr;                        { p = Bildspeicher drucken         }
      115: diskbild(false);                 { s = speichern Bildschirm         }
      108: diskbild(true);                  { l = laden Bildschirm             }
      116: textproc;                        { t = Textmodus ein                }
      102: fontsw:=not fontsw;              { f = font 8x8/8x14 ein/aus        }
      100: drawproc;                        { d = zeichnen ein/aus             }
      114: setrstproc;                      { r = Punkt setzen/löschen         }
       99: scrollproc;                      { c = Bildschirm scrollen          }
      120: xy:=not xy;                      { x = x/y vom Fadenkreuz anzeige   }
      118: vertext:=not vertext;            { v = text vertikal schreiben      }
      109: movebild;                        { m = Bild horizontal/vertikal     }
                                            { bewegen (rotieren).              }
       48: begin speed:=10;  display; end;  { 0 = 10 Punkte }
       49: begin speed:=1;   display; end;  { 1 =  1   "    }
       50: begin speed:=2;   display; end;  { 2 =  2   "    }
       51: begin speed:=3;   display; end;
       52: begin speed:=4;   display; end;
       53: begin speed:=5;   display; end;
       54: begin speed:=6;   display; end;
       55: begin speed:=7;   display; end;
       56: begin speed:=8;   display; end;
       57: begin speed:=9;   display; end;  { 9 =  9 Punkte }
       42: begin speed:=speed*10; display; end; { speed * = 10 Punkte }
       43: begin speed:=speed+1;  display; end; { speed + 1 }
       45: begin speed:=speed-1;  display; end; { speed - 1 }
       end;
until taste = 'E';                      { Taste "E" Programmende  }

   graphics(false);                     { Grafikmodus ausschalten }
end.
```

```
{------------------------------------------------------------------------------}
{ Read/Write Diskett zum/vom Bildschirmspeicher.                               }
{ Die var "bspeicher" (Bildschirmspeicher) ist auf eine Bildgröße von          }
{ 640 x 406 Bildpunkte oder = 80 x 406 = 32480 Bytes gesetzt.                  }
{ Note: Folgende Reihenfolge ist notwendig                                     }
{                 1. Procedur "speicherebeneread(ebene)"  vor                  }
{                    Procedur "savescreen(filename)".                          }
{                 2. Procedur "speicherebene(ebene)"      vor                  }
{                    Procedur "loadscreen(filename)".                          }
{------------------------------------------------------------------------------}
{ Filename: SaveLoad.EGA }

type namestring=string[12];
     bspeicher=array [0..32479] of byte;        { Bildspeicherhöhe 406 Zeilen }
     bildpointer=^bspeicher;                    { (406 * 80 = 32480 Bytes)    }

procedure SaveScreen(filename:namestring);
  type bildspeicher=file of bspeicher;
  var bildstart:bildpointer;
      bildfile:bildspeicher;

  begin
    bildstart:=ptr($A000,000);
    assign(bildfile,filename);
    {$I-} rewrite(bildfile); {$I+}
          if ioresult = 0
          then begin
               write(bildfile,bildstart^);
               close(bildfile);
               end
          else write('IO error file schreiben');
  end;
procedure LoadScreen(filename:namestring);
  type bildspeicher=file of bspeicher;
  var bildstart:bildpointer;
      bildfile:bildspeicher;
```

```
  begin
    bildstart:=ptr($A000,000);
    assign(bildfile,filename);
    {$I-} reset(bildfile); {$I+}
          if ioresult = 0
          then begin
               read(bildfile,bildstart^);
               close(bildfile);
               end
          else write('IO error file lesen');
  end;
{* Filename: Grafik.EGA *}
const
    egabase=$A000;                    { Startadresse des Bildschirmspeichers  }
var
   EGAmode                  :byte;    { 0F=Monochrome, 10=Enhanced Color Disp.}
   EGA64                    :boolean; { true=64k IBM EGA, false= < 64k IBM EGA}
   vert                     :integer; { vertikale Bildhöhe                    }
   savemode,startzeile      :integer; { Bildstart-Adresse                     }
   scrolline                :integer; { Scan line Länge in words oder Bytes   }
                                      { EGA 64k = 40, EGA > 64k = 80          }
   xtext,ytext              :integer; { Text x/y Position                     }
   lastxtext,lastytext      :integer; { letzte x/y Position                   }
   nextxy                   :boolean; { true, wenn noch kein Zeilenende       }
   wrtinvert,wrthor,graftext:boolean; { Text Schreibkontrolle                 }
   fontsegm,fontoffs        :integer; { Font pointer                          }
   conoutptrsave            :integer; { Turbo Pascal "ConOutPtr"              }
   pattern,box              :byte;    { bytes/charakter                       }
   i,j                      :integer;
{======================Grafik Text Write/ln Proceduren=======================}

{---------------------Font Pointer-------------------------------------------}
procedure Fontpointer(font:boolean);      { Diese Procedure ermittelt die     }
                                          { fontsegment und fontoffset Adresse}
                                          { von Font 8x8 und Font 8x14 im EGA }
                                          { ROM. }
type regs=record case integer of
            1:(ax,bx,cx,dx,bp,si,di,ds,es,flgs: integer);
            2:(al,ah,bl,bh,cl,ch,dl,dh: byte);
          end;
  var reg:regs;
begin
        reg.ax:=$1130;          { AH = EGA information              }
        if font
        then begin
             reg.bh:=3;         { 8x8 font }
             pattern:=8;        { 8 Bytes/charakter }
             box:=10;           { Box-Größe = 10 }
             end
        else begin
             reg.bh:=2;         { 8x14 font }
             pattern:=14;       { 14 Byte/Charakter }
             box:=14;           { Box-Größe = 14 }
             end;

        intr($10,reg);
        fontsegm:=reg.es;       { Font Segment-Adresse }
        fontoffs:=reg.bp;       { Font Offset-Adresse  }

end;

{-----------------------Turbo Pascal Gotoxy----------------------------------}
procedure turbogotoxy(x,y:integer);
begin
  Gotoxy(x,y);                  { Turbo Pascal Gotoxy }
end;
{-------------------------Grafik Gotoxy--------------------------------------}
procedure GotoXY(X,Y:integer);  { Diese Procedur GotoXY ersetzt die standart  }
                                { Procedur im Turbo pascal, wenn der EGAmode  }
                                { $0F oder $10 ist. Ist EGAmode 0, so wird    }
begin                           { über die Procedur "turbogotoxy" die Turbo   }
                                { Pascal Gotoxy aufgerufen.                   }

  if EGAmode = 0 then turbogotoxy(x,y); { => Turbo Gotoxy }

  xtext:=X;
  ytext:=Y;
end;
{--------------------------Write Text im Grafikmodus-------------------------}
procedure writechar(c :byte);         { Diese Procedure schreibt den Charakter }
var offset,i : integer;               { "c" an die Position x/y.               }
    inv      : byte;
begin
  if graftext
  then
      offset := xtext - 1 + (ytext * 80)                { Byte-Offset }
  else
      offset := xtext - 1 + ((ytext - 1) * (80*box));   { im Bildspeicher }

  if wrtinvert
  then
      inv:=$ff                      { invertiert alle charakter }
  else
      inv:=0;

if box = 10                         { Boxgröße 10 bei 8x8 Font }
then begin
     mem[egabase:offset]:= inv;     { + 1 blank/invert Kopfbyte/Charakter }
     offset:= offset+80;
     end;

  for i := 0 to pattern-1           { Charakter Font loop }
  do begin
     mem[egabase:offset]:=mem[fontsegm:fontoffs+(c*(pattern))+i] xor inv;
     offset:= offset + 80;          { nächste scan line }
  end;

if box = 10                         { Boxgröße 10 bei 8x8 Font }
then
     mem[egabase:offset]:= inv;     { + 1 blank/invert Fußbyte/Charakter }

end;
{----------------Start-Procedur für Turbo Pascal "write/ln"-------------------}
procedure writetext(c: byte);         { Diese Procedure ist die Startprocedure  }
                                      { von write(ln), im Grafik-Modus. Die x/y-}
                                      { Position, und die Kontrollcodes BS,LF und}
                                      { CR werden verarbeitet.                  }
                                      { Kein automatischer LF weder horizontal  }
                                      { noch vertikal.                          }
begin
  lastxtext:=xtext;                   { letzte x-Position      }
  lastytext:=ytext;                   { letzte y-Position      }
  case c of
{----------------BS (Backspace)---------------}
        8:   if wrthor
             then begin
                  if xtext > 1
                  then
                      xtext:=xtext - 1;             { <= 1 Charakter }
                  end
             else if graftext
                  then begin
                       if ytext >= box
                       then
                           ytext:=ytext - box;      { 1 Char.-Box nach oben }
                       end
                  else if ytext > 1
                       then
                           ytext:=ytext - 1;        { 1 Charakter nach oben }
{-----------------LF (Line feed)--------------}
```

```
        10: if not graftext
            then begin
                 if (box = 10) and (ytext < 35) or
                    (box = 14) and (ytext < 25)
                 then
                     ytext:=ytext + 1;                { nächste Zeile }
                 end;
{----------------CR (Carriage return)--------}
        13: xtext:= 1;                                { Spalte 1 }
{------------------alle Charakter------------}
  $00..$FF: begin
            {-----------}
            writechar(c);                             { Charakter schreiben }
            {-----------}
            if wrthor                                 { Text horizontal       }
            then begin
                 if xtext < 80
                 then
                     xtext:=xtext + 1;                { nächste x-Positin }
                 end
            else if graftext                          { Text vertikal,graftext }
                 then begin
                      if ytext <= (vert-(2*box)+1) { nicht über "vert" gehen}
                      then
                          ytext:=ytext + box;    { nächste y-Position, graftext}
                      end
                 else if (box = 10) and (ytext < 35) or
                         (box = 14) and (ytext < 25)
                      then                                { not graftext }
                          ytext:=ytext + 1;               { nächste y-Position }
            end;
  end; { case of c }

  if (lastxtext <> xtext) or (lastytext <> ytext)
  then
      nextxy:=true                                    { neue x/y-Position }
  else
      nextxy:=false;                                  { Zeilenende erreicht }
end;

{#################### I B M   E G A  Grafik-Hilfs-Procedure###################}

{----------------------Graphics Modus EIN/AUS schalten-------------------------}
procedure Graphics(on:boolean);

type regs=record case integer of
             1:(ax,bx,cx,dx,bp,si,di,ds,es,flgs: integer);
             2:(al,ah,bl,bh,cl,ch,dl,dh: byte);
           end;
   var reg:regs;
 begin
     if on                          { Grafiksmode einschalten }
     then begin
          reg.ax:=$0F00;            { AH = 0F lesen des EGA Status }
          intr($10,reg);
          savemode:=reg.al;         { EGA mode in AL }

          reg.ax:=$1200;            { Alternate select }
          reg.bl:=$10;              { EGA-Information }
          intr($10,reg);

          EGAmode:=0;

          if reg.bl = 0             { 0=64k, 1=128k, 2=192k, 3=256k EGA Memory }
          then begin
               EGA64:=true;
               scrolline:=40;       { 40 words = 80 Bytes (eine scan line) }
               end
          else begin
               EGA64:=false;
               scrolline:=80;       { 80 Bytes (eine scan line) }
               end;

          if reg.cl in [2,3,8,9]    { Enhanced Color Display ? }
          then
              EGAmode:=$10;
          if reg.cl in [4,5,10,11]  { Monochrome Display }
          then
              EGAmode:=$0F;

          if EGAmode  in [$0F,$10]
          then
               if EGAmode = $0F
               then
                    Write('EGA 640x350 Grafik, Monochrom Display')
               else
                    Write('EGA 640x350 Grafik, Enhanced Color Display')
          else begin
               write('EGA Grafik 640x350 mit dieser Grafik-Karte nicht möglich');
               Halt;        { Programm HALT !!!! }
               end;

          reg.ah:=$00;                    { EGA Modus }
          reg.al:=EGAmode;                { EGA Mode $0F/$10 setzen }
          intr($10,reg);                  { EGA BIOS Intrrupt $10 }
          conoutptrsave := conoutptr;     { Turbo Pascal conoutptr speichern }
          conoutptr := ofs(writetext);    { Grafik write/ln interface }
          wrtinvert := false;             { normal/reverse video (normal) }
          wrthor    := true;              { horizontal/vertikal schreiben (horiz.)}
          graftext:=true;                 { erlaube jede scan line (y-Achse)}
          fontpointer(true);              { true = 8x8 font im grafik text }
                                          { false = 8x14 font im grafik text }
          vert := 349;                    { verticale Bildhöhe = 349 scan lines,}
          end                             { wird im Userprogramm umdefiniert    }

          else begin                      { Grafiks Mode ausschalten }
          conoutptr:=conoutptrsave;       { zurück zu Turbo Pascal "write/ln"   }
          EGAmode:=0;
          reg.ah:=0;
          reg.ax:=savemode;               { zurück zu letztem EGA Modus }
          intr($10,reg);                  { EGA BIOS Interrupt $10 }
          end;
end;

{---------------------Pascal Punkt setzen-------------------------------------}
{ Punkt setzen an der Position x/y, Farbe = color, Funktion = funktion        }

procedure pset(x,y:integer;color,funktion:byte);
var offset   :integer;
    bitmask:byte;
begin

bitmask   := $80 shr (x and 7 );  { x-Bit Position im Byte }
offset    := (y * 80)+(x shr 3);

                                  { GK = Grafiks-Kontroler }

      port[$3ce] := 0;            { GK SET/RESET Reg. }
      port[$3cf] := color;        { Farbe ins SET/REST Reg. }
      port[$3ce] := 1;            { GK ENABLE SET/RESET Reg.}
      port[$3cf] := $0F;          { erlaube alle Planes }
      port[$3ce] := 3;            { GK Rotate Reg. }
      port[$3cf] := (funktion and $03) shl 3;
                                  { Funktion  00 unmodifiziert          }
                                  {           01 UND mit gelesenen Daten }
                                  {           10 OR "         "      "   }
                                  {           11 XOR "        "      "   }

      port[$3ce] := 8;            { GK BIT MASK Reg. }
      port[$3cf] := bitmask;      { PSET nur das durch x/y ausgewaehlte bit }

      mem[egabase:offset]:=mem[egabase:offset]; { read/write, siehe Text     }
                                                { ueber den Grafik Controller}
```

```
      port[$3cf] := $FF;          { erlaube wieder alle bits in GK Bit Mask Reg.}
      port[$3ce] := 1;            { GK ENABLE SET/RESET Reg. }
      port[$3cf] := 0;            { disable alle Planes }
      port[$3ce] := 3;            { GK Rotate Reg. }
      port[$3cf] := 0;            { Funktion 00    }
end;
{---------------------Bildschirm invertiern---------------------------------}

procedure Invertbild;        { 640x350 Bildschirm invertieren }
  begin
    port[$3ce]:=0;                      { GK Register 0                    }
    port[$3cf]:=15;                     { alle bits set/reset              }
    port[$3ce]:=1;                      { GK Register 1                    }
    port[$3cf]:=15;                     { erlaube alle bits                }
    port[$3ce]:=3;                      { GK Register 3                    }
    port[$3cf]:=$18;                    { XOR Funktion                     }
                                        { invertiere alle Bytes des ganzen }
    move(mem[egabase:0],mem[egabase:0],28000);
                                        { sichtbaren Bildschirmspeichers   }
    port[$3cf]:=0;                      { Funktion 00                      }
    port[$3ce]:=1;                      { GK Register 1                    }
    port[$3cf]:=0;                      { disable set/reset                }
  end;

{---------------------Farbpalettenregister setzen---------------------------}
procedure Farbpalette(farbe,palettereg:integer);
  var regs:record case integer of
             1:(ax,bx,cx,dx,bp,si,di,ds,es,flgs: integer);
             2:(al,ah,bl,bh,cl,ch,dl,dh: byte);
           end;
begin
  with regs do
    begin                       { Farbpaletten Register setzen }
      ax:=$1000;
      bl:=palettereg and $0F;
      bh:=farbe      and $3F;
      intr($10,regs);
    end;
end;

{---------------------Schreibmaske für Speicherebene/n----------------------}
{ Note: Nur in die hier selektierte/n Bildspeicherebene/n des Bildschirmspei- }
{       chers kann geschrieben werden.                                       }

procedure Speicherebene(ebene:byte);
begin
     port[$3c4]:=2;       { Sequencer Adress Register= Map Mask Reg }
     port[$3c5]:=ebene;   { Map Mask Register, Schreibmaske für Speicherebene/n}
end;
{-----------------------------------------------------------------------------}
Procedure EGAwritemode(mode:byte);  { EGA write mode, siehe Mode Register des }
                                    { Grafik-Kontrollers }
begin
     port[$3ce]:=5;       { Mode Register, Grafik-Kontroller }
     if EGA64
     then
         port[$3cf]:=$10 or mode    { ODD-Bit anlassen !!!! }
     else
         port[$3cf]:=mode;
end;
{---------------------Bildebene/n-Auswahl für Bildschirm--------------------}
{ Note: Nur die hier ausgewählte/n BildSpeicherebene/n werden auf dem Bild-  }
{       schirm angezeigt.                                                    }

procedure Bildebene(ebene:byte);
var a:byte;
begin
     if EGAmode = $0F
     then
          a := port[$3ba]   { Adressmodus, Attribute Kontroller (Monochrome)  }
     else
          a := port[$3da];  {      "          "          "       (Color     )  }

     port[$3c0] := $12;          { Color Plane Enable Register, Display AUS}
     port[$3c0] := ebene;        { Display Speicherebene/n                  }
     port[$3c0] := $20;          { Display EIN                              }
End;
{-----------------------------------------------------------------------------}
Procedure speicherebeneread(ebene:byte);  { Nur die hier ausgewählte Bild-    }
                                          { speicherebene kann gelesen werden.}
begin
    port[$3ce] := 4;             { GK Read Map Select Reg. }
    port[$3cf] := ebene;
end;
{-------------------Bildschirm nach oben/unten scrollen---------------------}
{ Note: Nur mit der "c" Taste, oder wenn setstart true ist kann diese Proce- }
{ dur wieder verlassen werden. "i" gibt an, wieviel scan lines auf einmal    }
{ pro Tastendruck scrollen. Siehe Kurzbeschreibung der Procedure }
procedure scroll(i:byte;setstart:boolean);
var    taste:char;
       x    :byte;
       addr :integer;
    ctrstart:integer;

begin
if EGAmode = $0F
then
    addr:=$3ba         { Input Status Register (Monochrome) }
else
    addr:=$3da;        { Input Status Register (Color     ) }

ctrstart := startzeile;
repeat
if not setstart
then begin
     read(kbd,taste);
        case ord(taste) of
          72 : ctrstart := ctrstart + (scrolline*i); { i * Zeilen aufwärts }
          80 : ctrstart := ctrstart - (scrolline*i); { i * Zeilen abwärts  }
        end;
     { scrollen nur zwischen scan line 0 und "vert" }
     if (ctrstart < 0) OR (ctrstart > (vert-349)*scrolline)
     then
         ctrstart := startzeile;
     end
else
    taste:='c';                 { wenn setstart true ist, nicht auf "c" warten.}

repeat                          { lese "Input Status Register" so lange, bis das }
     x:=port[addr] and $08      { Signal "Vertical Retrace" (Bildsychronisation) }
until x = $08;                  { true ist. Siehe Beshreibung der Procedur.      }

{ Delay(1); }                   { Siehe Beschreibung }

if EGAmode = $0F           { CRT Start Address Register setzen }
then begin                      { Monochrome-Modus }
     port[$3b4] := $0c;         { CRT Kontroller Register $0C }
     port[$3b5] := hi(ctrstart); { Start Adresse high }
     port[$3b4] := $0d;         { CRT,Kontroller Register $0D }
     port[$3b5] := lo(ctrstart); { Start Adresse low  }
     end
else begin                      { Color-Modus }
     port[$3d4] := $0c;         { CRT Kontroller Register $0C }
     port[$3d5] := hi(ctrstart); { Start Adresse high }
     port[$3d4] := $0d;         { CRT Kontroller Register $0D }
     port[$3d5] := lo(ctrstart); { Start Adresse low  }
     end;

startzeile := ctrstart;         { Start Zeile speichern }
until taste='c';            { Warteschleife, bis die "c" Taste gedrückt wird}
end;
```

HELP	Zeichnen mit der I B M E G A	HELP
Taste	Funktion	
h	Tastenbelegung Ein/Aus. HELP	
E	Programmende.	
c	Bildschirm nach oben/unten scrollen, (cursor) Ein/Aus.	
l / s	Bildschirm laden / speichern (Diskette).	
m	Bild auf dem Bildschirm rotieren, Ein/Aus. (cursor Tasten)	
p	Bild vom Bildschirm auf Grafikdrucker drucken.	
d	Funktion "Zeichnen" Ein/Aus.	
r	Punkt setzen/löschen (Pset/Preset), Ein/Aus.	
cursor links	Fadenkreuz nach links bewegen.	
cursor rechts	Fadenkreuz nach rechts bewegen.	
cursor oben	Fadenkreuz nach oben bewegen.	
cursor unten	Fadenkreuz nach unten bewegen.	
1..9,0,*,+,-	Geschwindigkeit des Fadenkreuzes, Punkte/cursor Taste.	
t	Textmodus einschalten. Textende mit CR-Taste. (Eingabe)	
f	Font-Umschaltung, 8x8 / 8x14. (hin und zurück)	
i	Font invertiert schreiben, Ein/Aus, z.B. A oder A	
v	Text vertikal schreiben, Ein/Aus.	
x	X/Y Position am linken Bildrand anzeigen, Ein/Aus.	

Bild 5. Hinweise zum Gebrauch von EGA-DRAW

*

Punkte * 10. Ist die Punktezahl z. B. 7, so wird durch die Taste „*" die Punktezahl auf 70 gesetzt (die maximale Punktzahl ist 255).

+/-

Punktezahl +1 / -1.

t

Textmodus einschalten. In der Mitte von Zeile 1 erscheint z. B. „Textmodus 8×8", d. h. Texteingabe horizontal. Es wird der 8×8-Font benutzt. Startposition des Textes ist die Position des Fadenkreuzes. Wird die Taste „t" gedrückt, so wird als erstes das kleine Fadenkreuz auf die nächste Bytegrenze nach links versetzt (wenn nötig). Das Fadenkeuz läuf dann als Cursor bei der Texteingabe mit. Bei der Texteingabe ist ebenfalls die BS-Taste (Back Space) wirksam, ohne daß aber bereits eingegebener Text gelöscht wird. Die Texteingabe wird mit „CR" beendet.

f

Umschalten zwischen Font 8×8 und 8×14.

i

Text invertiert (Reverse Video) schreiben.

v

vText vertikal schreiben. Die Taste BS ist auch hier wirksam. Sie bewegt den Cursor jetzt aber von unten nach oben.

x

xy-Position des Fadenkreuzes am linken Bildrand anzeigen (x horizontal, y vertikal). Bis auf die Funktion „zeichnen" erscheint der jeweilige Modus in der Mitte der Zeile 1. Zum Beispiel „Text 8×14 vertikal invertiert". In diesem Textmodus ist die Leer-Taste gut zu verwenden (dicke Linien, Mauern). Die Tasten „c,m,d,r,f,i,v und x" schalten die entsprechende Funktion hin und her.

Farben auf dem Bildschirm:

Beim Monochrom-Bildschirm werden das Fadenkreuz und die Programminformationen am Bildrand in Intensified Video und die Zeichnung in Normal Video dargestellt. Bei Überlappung von beiden entsteht Blinken zwischen Normal und Intensified Video. Dieses ist für die genaue Positionierung des Fadenkreuzes sehr wertvoll. Beim Enhanced-Color-Bildschirm ist die Farbe des Fadenkreuzes und der Programminformation „pink" und die Farbe der Zeichnung „rot". Bei Überlappung von beiden entsteht die Farbe „gelb".

Diskette:

Für die Speicherung eines Bildes werden auf der Diskette 32480 Bytes benötigt. Die *Bilder 6,7 und 8* zeigen, was man in recht kurzer Zeit auf den Bildschirm zaubern kann. Das Bild 8 zeigt das mit der Funktion „m" rotierte Bild 7. Es folgt die Kurzbeschreibung der wichtigsten EGA-DRAW-Prozeduren.

Procedure fadenkreuz(lr,ud:integer);

Diese Prozedur bewegt das Fadenkreuz, und zeichnet das Bild, wenn die Variable „draw" true ist. Parameter „lr": Nach links/rechts bewegen. -lr nach links, +lr nach rechts.„ud": Nach oben/unten bewegen. -ud nach oben, +ud nach unten.

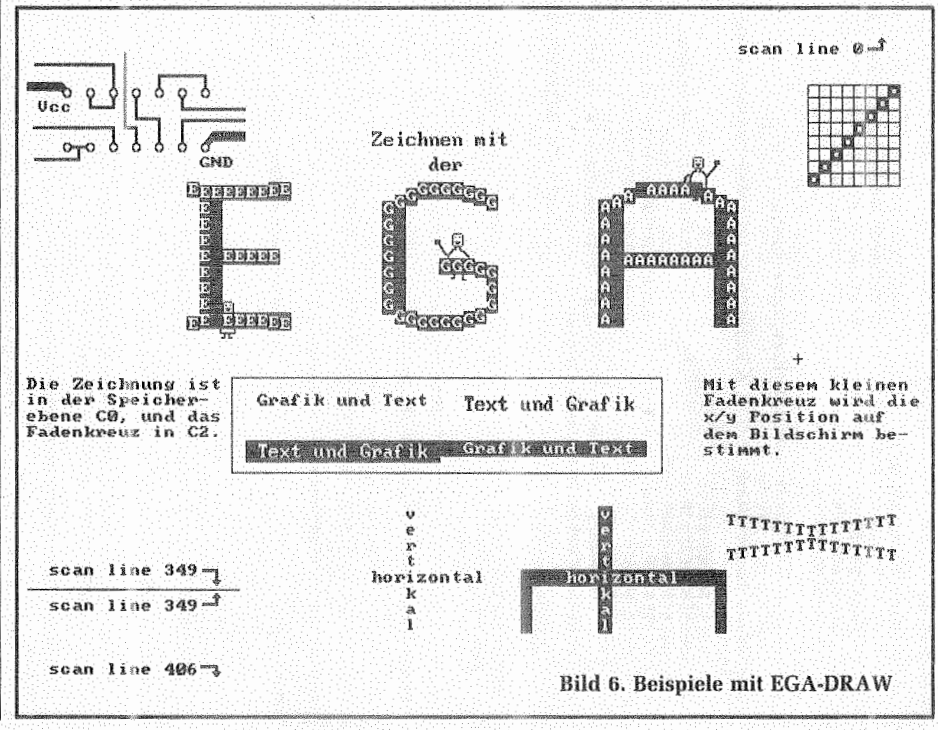


Bild 6. Beispiele mit EGA-DRAW

Procedure textproc;

Diese Prozedur kontrolliert die Funktionen Text: Vertikal, horizontal, 8×8-Font, 8×14-Font und Text invertieren. Ebenso wird die Bewegung des Cursors (Fadenkreuz) von dieser Prozedur gesteuert.

Procedure drawproc;

Diese Prozedur schaltet zeichnen/löschen ein oder aus.

Procedure setrstproc;

Diese Prozedur schaltet zwischen Zeichnen und Löschen hin und her.

Procedure diskbild(lesen:boolean);

Parameter „lesen": true = Bildfile von der Diskette lesen und in den Bildspeicher laden; false = Bildspeicher auf die Diskette schreiben.

Procedure movebild;

Mit Hilfe dieser Prozedur kann das Bild in der Zeichnungsebene sowohl horizontal als auch vertikal bewegt werden. Es wird in der entsprechenden Achse rotiert. Während beim Scrollen beide Speicherebenen (Fadenkreuz und Zeichnung) bewegt werden, bleibt bei der Prozedur „movebild" die Fadenkreuzebene stehen.

Procedure printscr(ebene);

Mit dieser Prozedur wird das Bild in der Zeichnungsebene auf dem Matrixdrukker ausgedruckt.

Procedure helpproc;

Diese Prozedur erlaubt es, zu jedem Zeitpunkt die Tastenbelegung auf dem Bildschirm anzuzeigen, ohne daß das Bild in der Zeichnungsebene berührt wird. Das Help-Bild wird immer von der Diskette gelesen. Es hat den Filenamen help.scr. Hat die IBM-EGA 128 KByte Speicher oder mehr, so kann das Help-Bild auch fest in eine der zwei noch freien Speicherebenen geladen werden.

Programm Start

Am Anfang des EGA-DRAW-Programmes wird untersucht, welche Ausrüstung die IBM-EGA-Karte hat und welcher Bildschirm angeschlossen ist. Enhanced Color Display:
Farbe pink = Fadenkreuz/Modus,
rot = Zeichnung
gelb = Überlappung von Zeichnung und Fadenkreuz/Modus

Monochrome Bildschirm:

Farbe Intensified Video = Fadenkreuz/Modus,
Normal Video = Zeichnung.
Blinking = Überlappung von Zeichnung und Fadenkreuz/Modus
An dieser Stelle sollte daran erinnert werden, daß bei EGA-Karten mit mehr als 64 KByte Speicher noch !!zwei!! Speicherebenen frei sind, die z. B. das Bild mit der Tastenbelegung und eine weitere Zeichnung – oder auch zwei weitere Zeichnungen aufnehmen könnte (zum Beispiel die Verdrahtung von zweiseitigen Leiterplatten). Weiterhin läßt sich bei 256-KByte-EGA-Karten die Vertikale dann noch mal verdoppeln, so daß in einer Bildspeicherebene von 640×817 Pixel gearbeitet werden kann.

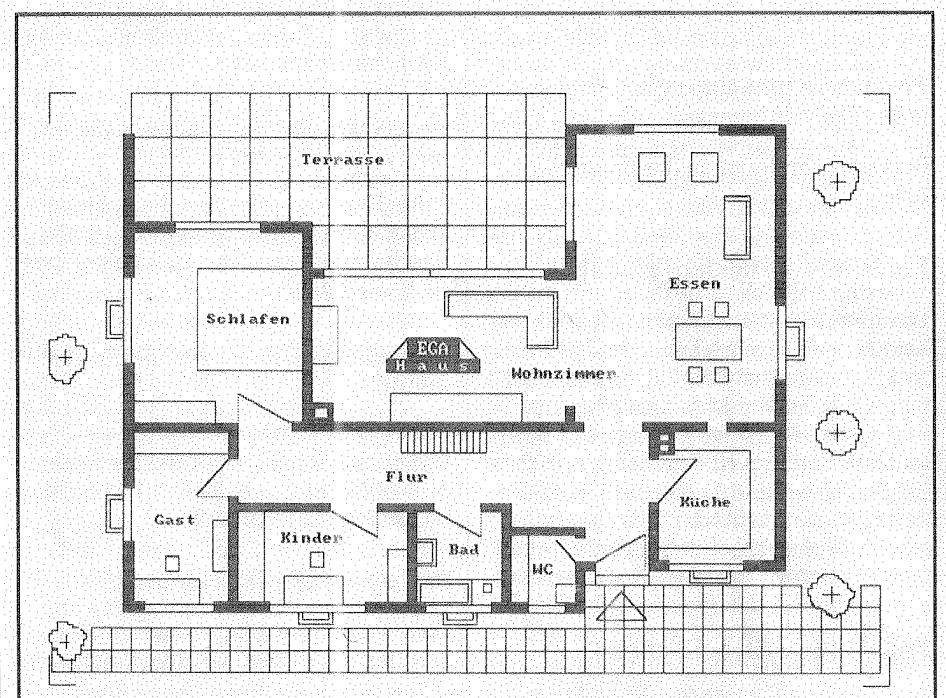


Bild 7. Ein Grundriß...

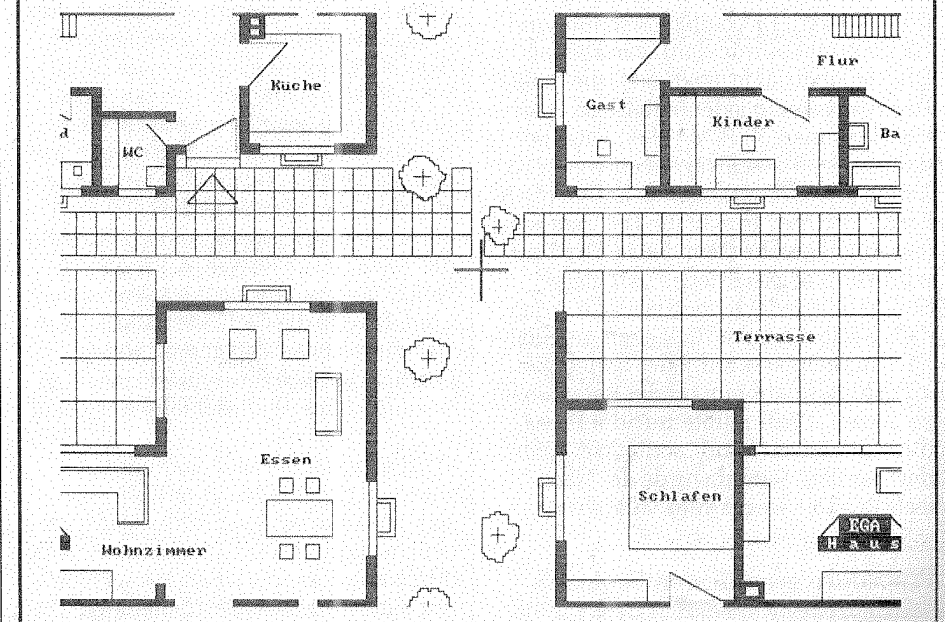


Bild 8. ...und gedreht

Die Programmänderungen sind minimal. Es folgt die Kurzbeschreibung der wichtigsten IBM-EGA-Prozeduren in der Include-Datei „Grafik.EGA".

Prozedure fontpointer(font:boolean);

Diese Prozedur ermittelt die Segment-/Offset-Adresse der beiden im EGA-ROM gespeicherten Charakter-Generatoren (Font). font true = 8×8-Font, font false = 8×14-Font.

Prozedure tubogotoxy(x,y:integer), gotoxy(x,y:integer);

Siehe Beschreibung in der Programmliste.

Prozedure writechar(c:byte);

Diese Prozedur ist eine Modifikation der bereits oben beschriebenen Pascal-Prozedur „writechar" (Bild 3), mit dem Unterschied, daß das Map Mask Register hier nicht benutzt wird. Writechar wird in der Prozedur writetext benutzt. Hat die Steuervariable „wrtinvert" den Inhalt true, dann wird der Text invertiert (reverse video) geschrieben. Beim Inhalt false wird der Text normal geschrieben.

Prozedure writetext(c:byte);

Diese Prozedur ist die Ersatz-Prozedur oder Start-Prozedur für alle „write/ln" (CON) Instruktionen, die normalerweise vom Turbo-Pascal ausgeführt werden. Hier werden auch die Kontrollcodes BS (Rücksetzen), LF (Zeilenvorschub) und CR (Schreibkopfrücklauf) verarbeitet. Die Steuervariable „graftext" true heißt: Der Text kann in jeder beliebigen Scan Line beginnen. Z. B. y = 388. False bedeutet: Der Bildschirm hat entweder 25 Zeilen (8×14-Font) oder 35 Zeilen (8×8-Font). „wrthor" true heißt: Der Text wird horizontal geschrieben. False: Der Text wird vertikal geschrieben. An dieser Stelle sollte jedoch noch gesagt werden, daß es selbstverständlich auch möglich ist, die Charakter auf jeder beliebigen x-Position (nicht nur auf der Bytegrenze) zu setzen. Dazu müßte nur die Procedure „writechar" und „writetext" geändert werden. Darauf wurde der Übersichtlichkeit wegen hier verzichtet.

Procedure graphics(on:boolean);

Diese Prozedur schaltet den Grafikmodus ein oder aus. Weiterhin wird untersucht, welche Ausrüstung die IBM-EGA besitzt (64k...256 KByte Speicher) und welcher Bildschirm angeschlossen ist. Folgende Kombinationen sind möglich: IBM-EGA 64k..256 KByte Speicher und Monochrome oder Enhanced Color Display. Falls die Prozedur eine andere Ausrüstung findet, erscheint bei der Ausführung des Programms die Nachricht: „EGA Grafik 640×350 ist mit dieser Grafik-Karte nicht möglich". Mit der Variablen „ConOutPtr" wird in dieser Prozedur die eigene write/ln-Prozedur „writetext" angeschlossen und der 8×8-Font wird selektiert.

Procedure speicherebeneread(ebene:byte);

Diese Prozedur selektiert die Speicherebene, die gelesen werden soll. (Siehe Beschreibung des Read Map Select Registers, Heft 2/87).

Procedure EGAwritemode(mode:byte);

Mit dieser Prozedur wird einer der 3 möglichen Schreibmodi ausgewählt. Siehe Beschreibung des Grafikkontroller Mode Register, Heft 2/87.

Procedure scroll(i,setstart);

Diese Procedure erlaubt das vertikale Scrollen des Bildes. Mit dem Parameter „i" kann die Geschwindigkeit des Scrollens bestimmt werden. Diese Prozedur ist die einzige, die zwei Register des CRT-Kontrollers setzt und zwar das Start Address High ($0C) und das Start Address Low ($0D) Register. Mit der Adresse in diesen beiden Registern beginnt der CRT Kontroller nach einem Bildwechsel den neuen Bildrefresh. Diese Routine ist gegenüber der in Heft 3 bereits beschriebenen leicht geändert. Mit dem Parameter „setstart" kann eine bestimmte Startadresse in die CRT-Startadress-Register gesetzt werden ohne auf die Taste „c" zu warten, die normalerweise das Scrollen beendet. Um zu verhindern, daß die CRT-Startadress-Register zur falschen Zeit gesetzt werden, (Vertical Retrace Bildsynchronisation) ist eine kleine Warteschleife eingebaut.

Wird nämlich die CRT-Startadresse während der Bildsynchronisation gesetzt, so kann es einen Bildwechsel geben, der gestört wird. Dieses macht sich durch eine kurze Bildverzerrung (von nur einem Bildwechsel) bemerkbar. Die kleine Warteschleife stellt sicher, das erst die Bildsychronisation beendet ist und dann die CRT-Startadress-Register gesetzt werden. Das ist aber nur eine Schönheitsmaßnahme. Die inaktive Instruktion „delay(1);" könnte bei schnelleren PCs, zum Beispiel beim AT, nötig sein.

Include File „SaveLoad.EGA"

Diese Datei enthält die beiden Unterprogramme „savescreen(filename)" und „loadscreen(filename)" mit denen das Bild auf die Diskette gespeichert oder von der Diskette gelesen werden kann. Das Array „bspeicher" ist fest auf die Bildspeichergröße 32480k Bytes (640×406) gesetzt. Will man andere Bildgrößen als 640×406 verwenden, so muß „bspeicher" entsprechend geändert werden.

Procedure savescreen(filename);

Bevor diese Prozedur aufgerufen wird, muß mit der Prozedur „speicherebeneread" die Bildspeicherebene selektiert werden, die auf die Diskette geschrieben werden soll.

Procedure loadscreen(filename);

Bevor diese Prozedur aufgerufen wird, muß mit der Prozedur „speicherebene" die Speicherebene selektiert werden, in die das Bild von der Diskette geladen werden soll.

Include „prtscr"

Diese Include-Datei enthält nur „prtscr(plane,vertikal)". Diese Prozedur wurde bereits in mc, Heft 3/87, kurz erklärt.

Zum Schluß möchte ich zusammenfassend sagen, daß, soweit es die grafische Anwendung betrifft, die wesentlichen Funktionen der IBM-EGA aufgezeigt wurden. Daß auch Horizontal Scrolling in Verbindung mit „Pel Panning" (siehe Grafik Kontroller) möglich ist, soll nicht unerwähnt bleiben – von den Tricks mal ganz abgesehen (mehr als 120 Character pro Zeile, externer Oscillator, usw). Und einen Texmodus gibt es auch noch, der sicher noch hoch interressant ist. Beim Verfasser läuft auch die „Turbo Graphix Toolbox" der Firma Borland mit „voller" Unterstützung der IBM-EGA, sowohl mit dem Monochrome-Bildschirm, EGA-Modus x'0F' APA 640×350, als auch mit dem Enhanced-Color-Bildschirm, EGA-Modus x'10' APA 640×350 mit allen Farben!

Ulrich Rohde

68020 im PC

Sie können den IBM-PC und seine Verwandten mit einer Einsteckkarte zum „Number Cruncher" machen. Die Firma SIG-Computer hat den 68020 zusammen mit einem 68881-Arithmetikprozessor und zwei MByte RAM (in SIP-Modulen) auf eine Karte gebracht.

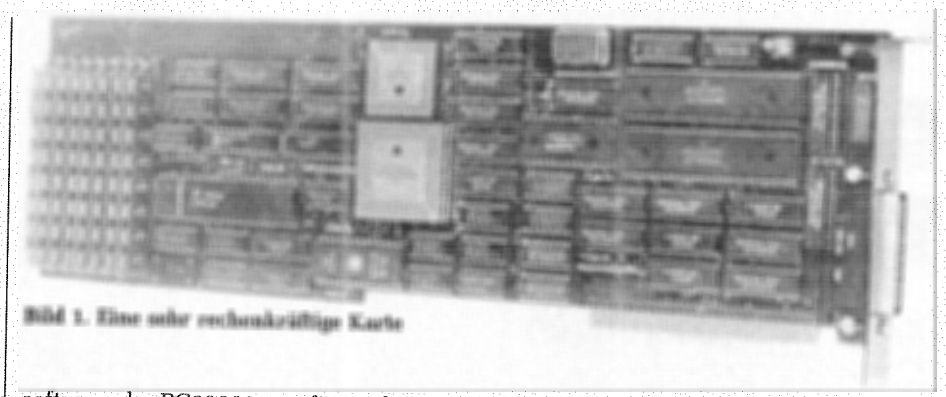


Bild 1. Eine sehr rechenkräftige Karte

Das *Bild 1* zeigt die wirklich beeindruckende Karte, die allerdings so viel Strom zieht, daß sie nicht mehr in jeden Billig-PC paßt, weil sie dessen Netzteil überlasten könnte. In der Redaktion wurde sie in einen M24 von Olivetti gesteckt, in dem sie klaglos ihren Dienst versieht.
Die Karte besitzt einen Bereich mit Dual-Port-RAM, über den sie mit dem PC kommuniziert. Auf der Karte PC68020 (so der offizielle Name) gibt es 32 KByte EPROM mit den grundlegenden System-Routinen. Die PC68020 fühlt sich zunächst als Slave-Rechner und kann vom PC aus über die Portadresse 300 jederzeit unterbrochen werden. Die Systemsoftware der PC68020 empfängt über dieses Port Kommandos, die ein Monitor dann absolviert. Obwohl man damit von Hand arbeiten könnte, sei jetzt gleich gesagt, daß zur mitgelieferten Software ein Emulator auf der PC-Seite gehört, der der Karte die Ressourcen des PCs so zur Verfügung stellt, daß man CP/M-68k darauf fahren kann. In *Bild 2* ist ein 68020-Maschinenprogramm mit 68881-Ansprache abgebildet, das den Kern einer Apfelmännchen-Iteration bildet. Es wurde von Michael Frank von Hand codiert, damit die Ansprache des 68881 auch regelrecht erfolgt. Wir hatten als Softwarepaket das Silicon-Valley-Pascal-System zur Verfügung, das über mehrere Durchgänge unter CP/M-68k (mit Compilieren, Objekt-Code Erzeugen, Hinzulinken der benötigten Routinen aus der C- und 68000-Bibliothek, Relocieren ...) letztlich ein unter CP/M-68k lauffähiges Programm erarbeitet. Das Programm in Bild 2 erfüllt die Konventionen dieses Systems und kann von Pascal aus als externe Funktion verwendet werden.
Seine Laufzeit war etwa um den Faktor 8 kürzer, als die Laufzeit, die wir gestoppt haben, als ein reines, in der Logik äquivalentes Pascalprogramm ohne 68881-Ansprache tätig war.
Die Zeiten: Etwa 19 KByte von Pascal erzeugter, lauffähiger Maschinencode wurde in 12,5 s von einer Diskette geladen. Die Apfelmännchenberechnung mit dem abgebildeten Maschinenprogramm innerhalb eines Pascalprogrammes dauerte 2 min 34 s. Dabei wurden in einem Gebiet (0 <= x <= 2, 0 <= y <= 1,09) für 350 CRT-Zeilen zu je 640 Pixel je 100 Iterationsschritte maximal durchgeführt. In reinem Pascal (mit der logisch äquivalenten Pascaliteration) dauerte diese Rechnung 16 min 28 s.
Die Einsteck-Karte kann einerseits als eigenständiges System benutzt werden, aber auch als Parallelrechner zum PC, der alle schwierigen Rechenaufgaben übernimmt und selbstständig löst. Man kann nämlich das DOS-Emulat auf dem PC einschläfern, sodaß es nicht mehr im Abfragebetrieb auf den 68020 wartet. Die Karte wird mit 14 MHz getaktet. Sie kostet etwa 5000 DM.

```
.globl TESTAPFEL
.text
*
* Registers used:
*    x: fp1
*    y: fp2
*    xn: fp3
*    i: d0
*    temp1: fp4
*    temp2: fp5
*
* function testapfel(imax: integer;
* xc,yc:real):integer;
TESTAPFEL:
     link a5,#-4
*
*    i:=0;
*    x:=xc; y:=yc;
*
     moveq.l #0,d0
*     fmove.s 12(a5),fp1
     dc.w $f22d,$4480,$000c
*     fmove.s 8(a5),fp2
     dc.w $f22d,$4500,$0008
*     while (sqr(x)+sqr(y) <=4.0)
*     and (i<imax) do

LABEL1:
*     fmove.s fp1,fp4
     dc.w $f200,$0600
*     fmove.s fp2,fp5
     dc.w $f200,$0a80
*     fmul.s fp4,fp4
     dc.w $f200,$1223
*     fmul.s fp5,fp5
     dc.w $f200,$16a3
*     fmove.s fp5,fp6
     dc.w $f200,$1700
*     fadd.s fp4,fp6
     dc.w $f200,$1322
*     fcmp.w #4,fp6
     dc.w $f23c,$5338,$0004
*     fbge.s LABEL2
     dc.w $f283
     dc.w LABEL2-*
*
     cmp.w 16(a5),d0
     beq.s LABEL2
*     i:=i+1;
     addq.l #1,d0
*      xn:= sqr(x) - sqr (y) + xc ;
*     fsub.s fp5,fp4
     dc.w $f200,$1628
*     fadd.s 12(a5),fp4
     dc.w $f22d,$4622,$000c
*      y  := 2 * x * y + yc;
*     fmul.s fp1,fp2
     dc.w $f200,$0523
*     fadd.s fp2,fp2
     dc.w $f200,$0922
*     fadd.s 8(a5),fp2
     dc.w $f22d,$4522,$0008
*      x  := xn;
*     fmove.s fp4,fp1
     dc.w $f200,$1080
     bra LABEL1
*
*    if i=imax then testapfel := 0 else
*    testapfel:= 1 + (i mod 15);
*
LABEL2:
     cmp.w 16(a5),d0
     bne LABEL3
     moveq.l #0,d0
     bra.s LABEL4
LABEL3:
     and.l #15,d0
     cmp.b #15,d0
     beq.s LABEL4
     addq.l #1,d0
LABEL4:
     unlk a5
     move.l (sp)+,a0
     lea.l 10(sp),sp
     jmp (a0)
*
     end
```

Bild 2. Von Hand codierte Apfelmännchen-Iteration

Christian Philipps

Command unter Turbo-Pascal

Einige Verbesserungen

In dem Artikel „Turbo ruft Command", der in der mc 11/86 erschien, wurde gezeigt, wie in Turbo-Pascal eine Kopie des DOS-Kommandoprozessors geladen und zur Ausführung z. B. interner DOS-Befehle verwendet werden kann. Mit einigen Änderungen funktioniert das Programm fehlerfrei.

Wenn auch das vorgestellte Programm in der abgedruckten Form tatsächlich das Inhaltsverzeichnis des angemeldeten Laufwerkes auf den Bildschirm zaubert und anschließend scheinbar korrekt terminiert, so enthält es doch einen gravierenden Denkfehler: Bei meinen eigenen Experimenten in dieser Richtung versuchte ich ebenfalls zunächst einmal, die Standardfunktion MSDOS für den Aufruf der DOS-Exec-Funktion zu verwenden. Nachdem einige Versuche, bei denen der gesamte Aufrufmechanismus im Hauptprogramm untergebracht war, erfolgreich verlaufen waren, beschloß ich, ähnlich der vorgestellten Prozedur DOS, den Aufrufkomplex als Pascal-Prozedur zu realisieren, die ich dann in andere Programme einzubinden gedachte.

Gesagt, getan – die neue Prozedur (*Bild 1*) erhielt den Namen EXEC und hatte sogleich Premiere in einem weiteren kleinen Testprogramm. Erstaunlicherweise ergaben sich nun Schwierigkeiten, die zuvor nie aufgetreten waren; der Prozeduraufruf und das Laden des Command wurde augenscheinlich ordnungsgemäß durchgeführt, denn das Inhaltsverzeichnis meiner Diskette wurde auf dem Bildschirm angezeigt. Als letzte Anweisung hatte ich hinter den Aufruf der Prozedur EXEC noch eine WRITELN-Anweisung geschrieben, die mir, sozusagen zur Befriedigung des Ego, eine Erfolgsmeldung ausgeben sollte, bevor das Programm endete. – ... es geschah nichts dergleichen. Einige eingestreute WRITELN-Anweisung brachten dann auch bald die Stelle ans Tageslicht, an der der Fehler lag: Alle Anweisungen, die sich vor dem Aufruf von EXEC im Hauptprogramm befanden und alle Anweisungen innerhalb der Prozedur wurden korrekt durchlaufen. Beim Rücksprung aus EXEC ins Hauptprogramm, jedoch, brach das Programm entweder kommentarlos ab, mit der Folge, daß ich mich auf der DOS-Ebene wiederfand (und in der Regel die DOS-Speicherstruktur zerstört war), oder aber der Rechner hängte sich auf. Offenbar war also durch den Aufruf des Command etwas im Rücksprung-Mechanismus des Turbo-Pascal-Programmes durcheinandergeraten, d.h. der Stack mußte auf irgendeine Art und Weise Schaden genommen haben. Ein Studium des entsprechenden Abschnittes im technischen DOS-Handbuch brachte mir das heißersehnte Aha-Erlebnis: Laut Auskunft dieses Handbuches zerstört die Funktion $4B des Interrupt $21 sämtliche Register, also auch die Register SS, SP und BP, die für die Stackverwaltung in Turbo-Pascal lebenswichtig sind und von (meines Wissens nach) allen anderen Funktionen des Interrupt $21 nicht angetastet werden. Die ach-so-praktische Standardfunktion MSDOS braucht sich also im Regelfall nicht um die Ret-

```
PROGRAM Ex;

TYPE  RegType = RECORD CASE Integer OF
                  1: (ax,bx,cx,dx,bp,si,di,ds,es,flags : INTEGER);
                  2: (al,ah,bl,bh,cl,ch,dl,dh          : BYTE);
                END;

CONST Regs : RegType = (ax:0;bx:0;cx:0;dx:0;bp:0;si:0;di:0;ds:0;es:0;flags:0);

VAR   Cmd  : STRING[128];

{////////////////////////////////////////////////////////////////////////////////
///                                                                            ///
///             T U R B O  -  P A S C A L  Execute - Toolbox                   ///
///               (c) Copyright Okt. 1986 by C.Philipps                        ///
///                                                                            ///
///                    (Turbo Pascal V3.0 required)                            ///
///                                                                            ///
////////////////////////////////////////////////////////////////////////////////
///                                                                            ///
///  Die Routinen dieser Toolbox setzen eine Registerstruktur REGS (siehe      ///
///  auch Tubo-Pascal Handbuch) als CONST-Deklaration im Hauptprogramm de-     ///
///  finiert voraus.                                                           ///
///                                                                            ///
////////////////////////////////////////////////////////////////////////////////}

PROCEDURE FreeMemoryNotUsed;

CONST  Turbo_SP : INTEGER = 0;

{ Die Prozedur FreeMemoryNotUsed benutzt die DOS-Funktion SETBLOCK ($4A)
  um den oberhalb des TURBO-Stack-Segmentes gelegenen Speicherbereich an
  den DOS-Memory-Manager zurückzugeben. Der Aufruf dieser Prozedur sollte
  die erste Anweisung im Hauptprogramm sein, da sie sich bei der Ermittlung
  des freizugebenden Speichers auf den gegenwärtigen Wert des SP-Registers
  bezieht; anderenfalls könnten Stack-Daten verlorengehen. }

BEGIN  {FreeMemoryNotUsed}
  INLINE($2E/$89/$26/Turbo_SP);          { MOV CS:Turbo_SP,SP     }
  IF Turbo_SP < 0                        { Stacksegment > 64K     }
     THEN BEGIN
            Writeln(^G'Fehler bei SETBLOCK! - (Turbo_SP < 0)');
            HALT(1);                     { Notstop und Errorlevel setzen }
          END;
  WITH Regs DO
  BEGIN
    ES  = CSeg;
    BX  =(SSEG-CSEG)+1+((Turbo_SP+22) DIV 16); { Anzahl für Turbo benötigter
                                                 Paragraphen errechnen und...}
    AH  = $4A;                                 { ... mit der DOS-Funktion
                                                 SETBLOCK reservieren.      }
    MSDOS(Regs);
    IF (FLAGS AND 1) = 1
```

Bild 1. Die verbesserte Version funktioniert jetzt fehlerfrei

```
      THEN BEGIN
             Writeln(^G'Fehler bei SETBLOCK: AH=',AH,' AL=',AL);
             HALT(1);
           END;
  END;
END;   {FreeMemoryNotUsed}

{-------------------------------------------------------------------------------}

FUNCTION Exec(VAR CommandLine):Byte;

{ Diese Funktion läd eine zweite Kopie des DOS-COMMAND.COM und übergibt
  ihm die Commandline, die in der gleichnamigen Variable abgespeichert ist.
  Falls ein Fehler auftritt, wird der Inhalt des AX-Registers an den aufru-
  fenden Prozess zurückgegeben, anderenfalls ist der Returncode 0.

  Laut DOS-Handbuch können bei der Funktion $4B des Interrupt $21 folgende
  Fehler auftreten:

  1: AL ist nicht 0 oder 3
  2: Programm nicht gefunden oder ungültiger Pfadname
  8: Nicht genug Speicher zum Laden des Programmes                          }

CONST  ZSS    : INTEGER = 0;                    { Zwischenspeicher für Register }
       ZSP    : INTEGER = 0;
       ZBP    : INTEGER = 0;
       Path   : STRING[$80] = '\COMMAND.COM'#0; { sucht auf dem angemeldeten
                                                  Laufwerk nach COMMAND.COM }

VAR    Parameterblock : RECORD
                          OffsetEnvironment   : INTEGER;
                          OffsetCommandLine   : INTEGER;
                          SegmentCommandLine  : INTEGER;
                          OffsetFCB1          : INTEGER;
                          SegmentFCB1         : INTEGER;
                          OffsetFCB2          : INTEGER;
                          SegmentFCB2         : INTEGER;
                        END;


BEGIN  {Exec}
  WITH Parameterblock DO
  BEGIN
    OffsetEnvironment  := 0;                { Umgebung vom aufrufenden Pro-
                                              gramm übernehmen }
    OffsetCommandLine  := Ofs(CommandLine);
    SegmentCommandLine := Seg(CommandLine);
    OffsetFCB1         := $5C;              { FCBs vom aufrufenden Programm
                                              kopieren; stehen ab Offset
                                              $5C und $6C im PSP }
    SegmentFCB1        := CSeg;
    OffsetFCB2         := $6C;
    SegmentFCB2        := CSeg;
  END;
  WITH Regs DO
  BEGIN
    DS := Seg(Path);
    DX := Ofs(Path)+1;                      { Längenbyte überspringen }
    ES := Seg(Parameterblock);
    BX := Ofs(Parameterblock);
    AL := 0;
    AH := $4B;
  END;
  INLINE ($50/                          { PUSH AX                 }
          $53/                          { PUSH BX                 }
          $51/                          { PUSH CX                 }
          $52/                          { PUSH DX                 }
          $06/                          { PUSH ES                 }
          $1E/                          { PUSH DS                 }
          $57/                          { PUSH DI                 }
          $56/                          { PUSH SI                 }
          $2E/$8C/$16/ZSS/              { MOV CS:ZSS,SS           }
          $2E/$89/$26/ZSP/              { MOV CS:ZSP,SP           }
          $2E/$89/$2E/ZBP/              { MOV CS:ZBP,BP           }
          $BB/REGS/                     { MOV BX,OFFSET REGS      }
          $2E/$8E/$5F/$0E/              { MOV DS,CS:[BX]0E        }
          $2E/$8B/$57/$06/              { MOV DX,CS:[BX]06        }
          $2E/$8E/$47/$10/              { MOV ES,CS:[BX]10        }
          $2E/$8B/$47/$00/              { MOV AX,CS:[BX]00        }
          $2E/$8B/$5F/$02/              { MOV BX,CS:[BX]02        }
          $CD/$21/                      { INT  21H                }
          $FA/                          { CLI                     }
          $2E/$8E/$16/ZSS/              { MOV SS,CS:ZSS           }
          $2E/$8B/$26/ZSP/              { MOV SP,CS:ZSP           }
          $2E/$8B/$2E/ZBP/              { MOV BP,CS:ZBP           }
          $FB/                          { STI                     }
          $BB/REGS/                     { MOV BX,OFFSET REGS      }
          $2E/$89/$47/$00/              { MOV CS:[BX]00,AX        }
          $9C/                          { PUSHF                   }
          $58/                          { POP AX                  }
          $2E/$89/$47/$12/              { MOV CS:[BX]12,AX        }
          $5E/                          { POP SI                  }
          $5F/                          { POP DI                  }
          $1F/                          { POP DS                  }
          $07/                          { POP ES                  }
          $5A/                          { POP DX                  }
          $59/                          { POP CX                  }
          $5B/                          { POP BX                  }
          $58);                         { POP AX                  }
    IF (Regs.FLAGS AND 1) = 1           { Carry-Flag gesetzt ?    }
      THEN Exec := Regs.AX              { Ja, dann Fehlercode in AX }
      ELSE Exec := 0;
END;   {Exec}

{-------------------------------------------------------------------------------}

PROCEDURE ExecError(ErrNo:BYTE);

{ Die Prozedur ExecError kann verwendet werden, um den von der Prozedur
  Exec gelieferten Returncode in eine Klartext-Fehlermeldung umzuwandeln.

  Ist ein Fehler aufgetreten, so wird eine entsprechende Meldung ausgegeben
  und das Programm sicherheitshalber Not-gebremst.

  Aufruf: ExecError(Exec(Cmdline));
}

BEGIN
  IF ErrNo = 0          { Wenn kein Fehler, dann sofort zurück }
    THEN Exit;
  CASE ErrNo OF
    1: Writeln(^G'Das AL-Register enthielt nicht 00H oder 03H!');
    2: Writeln(^G'Programm nicht gefunden oder ungültiger Pfadname!');
    8: Writeln(^G'Nicht genug Speicher zum Laden des Programmes!');
  ELSE Writeln(^G'Ein undefinierter Fehler ist aufgetreten! - AX=',ErrNo);
  END;
  Halt(1);              { .. ansonsten Nothalt mit Setzen des Errorlevels }
END;

{///////////////////////////////////////////////////////////////////////////////
///                          E N D   O F   M O D U L E                        ///
///////////////////////////////////////////////////////////////////////////////}

BEGIN
 FreeMemoryNotUsed;
 Cmd := '/C DIR'#13;
 ExecError(Exec(Cmd));
 Writeln('Exec-Aufruf beendet und hier bin ich wieder!!');
END.
```

tung dieser Register zu kümmern und tut dies scheinbar auch nicht, denn nachdem ich den gesamten Registerübergabe- und Aufrufmechanismus in ein INLINE-Statement verpackt hatte, arbeitete die Prozedur einwandfrei. Die kritischen Register werden vor dem INT 21-Befehl in den typisierten Turbo-Konstanten ZSS, ZSP und ZBP abgelegt und sofort nach der Rückkehr vom Interrupt wieder von dort zurückgeholt. Da, wie von MSDOS gewohnt, auch die Werte der Register wieder in die Turbo-Registerstruktur REGS übertragen werden, kann die gesamte Fehlerprüfung anschließend in Pascal abgewickelt werden. Zu diesem Zweck wurde aus der Prozedur EXEC die Funktion EXEC, die als Funktionswert einen Returncode 0, für O.K. oder einen Returncode entsprechend den im DOS-Handbuch dokumentierten Fehlercodes (AX-Register) zurückgibt. Zum Abschluß vielleicht noch ein Wort zu den von mir hinzugefügten Prozeduren ExecError und FreeMemoryNotUsed. ExecError dient lediglich dazu, wahlweise eine dem Returncode von EXEC entspechende Fehlermeldung auszugeben; dies würde durch einen Aufruf in der folgenden Form erreicht: ExecError(EXEC(parameter));. Die Prozedur FreeMemoryNotUsed dient dazu, explizit den oberhalb des Turbo-Pascal Stacksegments gelegenen Speicher an den DOS-Memory-Manager zurückzugeben. (Ob dies unbedingt erforderlich ist...? – Nun, es schadet jedenfalls nicht). Der Aufruf dieser Prozedur sollte die erste Anweisung im Hauptprogramm sein, da bei der Berechnung des benötigten Speicherplatzes Bezug auf den gegenwärtigen Inhalt des SP-Registers genommen wird. Haben sich zum Zeitpunkt des Aufrufes von FreeMemoryNotUsed bereits Daten auf dem Stack angesammelt, so könnte dies zu einem teilweisen Verlust dieser Daten führen.

Graphix-Toolbox und Fehlerbehandlung

Die Turbo-Graphix-Toolbox von Borland für den PC ist ein recht brauchbares Hilfsmittel bei der Erstellung von Grafiken. Die Behandlung von Fehlern, die außerhalb der Toolbox auftreten, ist jedoch sowohl im Handbuch der Toolbox als auch im Handbuch von Turbo-Pascal selbst etwas zu kurz gekommen. Jeder Programmabsturz durch einen nicht abgefangenen Fehler ist für sich alleine schon sehr unangenehm. Noch ärgerlicher wird es jedoch, wenn dieser Fehler im Grafikmodus auftritt. In diesem Fall erscheint die Fehlermeldung mit zugehöriger Fehleradresse als ein wirres Punktmuster innerhalb der Grafik und die Bildschirmkarte verbleibt im Grafikmodus. Für die Zurückschaltung in den Textmodus empfiehlt das Handbuch der Toolbox nach Programmende den prophylaktischen Aufruf eines mitgelieferten Programms namens HFIX. Die Zurückschaltung in den Textmodus läßt sich jedoch einfacher und eleganter lösen.

```
Type Str79 = String [79];

Procedure Own_Error_Handler (Number, A : Integer);

Const H : Array [0..15] of Char = '0123456789ABCDEF'; (* H-Zeichen *)

  Procedure Cent_Line (Y : Integer; Message : Str79);

  Begin
    GotoXY ((80 - Length (Message)) div 2, Y); (* Cursor setzen *)
    Write (Message) (* Text ausgeben *)
  End;

Begin

  (* Bei der Verwendung der Graphix-Toolbox würde man an
     dieser Stelle zunächst die Prozedur LeaveGraphic für
     Umschaltung in den Textmodus aufrufen. *)

  ClrScr; (* Bildschirm löschen *)
  TextBackGround (White); (* heller Hintergrund *)
  TextColor (Black); (* dunkle Schrift *)
  Cent_Line (5, '                                          ');
  Cent_Line (6, '  Das Programm ist leider wegen eines  ');
  Case Hi (Number) of (* Fehlerart ? *)
    1: Cent_Line (7, '    Ein-/Ausgabefehlers abgestürzt!    ');
    2: Cent_Line (7, '      Laufzeitfehlers abgestürzt!      ')
  else
    Cent_Line (7, '    unbekannten Fehlers abgestürzt!    ')
  End;
  Cent_Line (8, '                                          ');
  TextBackGround (Black); (* dunkler Hintergrund *)
  TextColor (White); (* helle Schrift *)
  GotoXY (29, 11); (* Fehlernummer ausgeben *)
  Write ('Fehlernummer:    ', H [Lo (Number) shr 4], H [Lo (Number) and $0F]);
  GotoXY (29, 12); (* Fehleradresse ausgeben *)
  Write ('Fehleradresse: ');
  Write (H [Hi(A) shr 4], H [Hi(A) and $F], H [Lo(A) shr 4], H [Lo(A) and $F]);
  GotoXY (1,20); (* Cursor ans Bildende setzen *)
  Halt
End;

Begin
  ErrorPtr := ofs (Own_Error_Handler); (* Fehlerroutine umlegen *)
  Writeln (5 / 0); (* Division durch Null erzeugt einen Run-Time-Error *)
End.
```

Mit einer kurzen Routine werden Fehler im Grafik-Modus aufgefangen

Im Normalfall ruft Turbo-Pascal nach einem fatalen Fehler eine eingebaute Fehlerroutine auf. Die Standardvariable ErrorPtr zeigt dabei auf die Adresse der Fehlerroutine. Weist man nun dieser Variablen die Adresse einer eigenen Prozedur zu, so erfolgt im Falle eines Fehlers ein Aufruf dieser Prozedur. Die Prozedur muß zwei Aufrufparameter vom Typ Integer enthalten. Das höherwertige Byte des ersten Parameters enthält die Fehlerart (z.B. 1 bei einem E/A-Fehler), während das niederwertige Byte die Fehlernummer enthält (entsprechend dem Handbuch). Der zweite Parameter enthält die Fehleradresse.

Eine solche Fehlerroutine sollte neben der Ausgabe von Art, Nummer und Adresse des Fehlers (möglichst in sedezimaler Form) auch die Zurückschaltung in den Textmodus veranlassen. Das im *Bild* gezeigte Programm enthält eine solche Fehlerprozedur und erzeugt mit der Division durch Null einen Laufzeitfehler.

Günther Sternberg

Tuende Kriegl

MS-DOS merkt sich 128 Befehle

Emulation der Unix-C-Shell

MS-DOS erinnert sich ausschließlich an den zuletzt eingegebenen Befehl. Alles, was davor eingegeben wurde, ist für immer vergessen. Besser haben es da Unix-Benutzer. Ihr Betriebssystem merkt sich eine Anzahl von Befehlen, die sich frei vereinbaren lassen. Ein C-Programm verleiht MS-DOS diese Eigenschaft.

Das im *Bild* abgedruckte Programm simuliert die Unix-C-Shell. Sie stellt für die 128 zuletzt eingegebenen Befehle folgende Funktionen zur Verfügung:

– Auflisten der Befehle mit *hi* (History-Befehl).

– Einen beliebigen Befehl nochmals ausführen lassen, entweder unverändert oder modifiziert (editiert).

– Die Eingabe *!n* startet den n-ten Befehl.

Mit *^string1^string2* kann man den zuletzt eingegebenen Befehl noch einmal ausführen, aber vor der Ausführung wird im Befehl *string 1* durch *string2* ersetzt.

Nach der Eingabe *!!* wird der zuletzt eingegebene Befehl unverändert ausgeführt.

!string sucht nach dem zuletzt eingegebenen Befehl, der mit *string* anfängt, und führt ihn aus.

!n:/string1/string2 führt den n-ten Befehl aus, ersetzt aber *string1* durch *string2*.

Den aktuellen Pfad zeigt man mit *pwd* an (Unix-Befehl); F3, Pfeil, Insert, Delete, Backspace arbeiten wie gewohnt. Das Programm meldet sich nach dem Starten mit dem $-Zeichen.
Beispiel:

```
A\dir1\dir2>history
$dir
--- MS-DOS-Directory wird ausgedruckt ---
$hi
0 dir
1 type c.b
$!0
--- MS-DOS Directory wird ausgedruckt --
$pwd
\dir1\dir2
$
```

Das Programm wurde mit dem Aztec-C-Compiler geschrieben und ist etwa 7 KByte (exe-File) lang. Der einzige Aztec-spezifische Aufruf ist pwd_proc (die letzte Subroutine). Er ruft die MS-DOS-Funktion „Return Text of Current Directory" auf (hex. 47). Die Subroutine „dos" heißt bei Lattice und Microsoft „intdos" und benötigt andere Parameter. Falls man sie mit dem Dummy *pwd proc(){}* ersetzt, funktioniert alles bis auf den pwd-Befehl.

Eine Einschränkung ist zu beachten: Environment-Modifikationen können nicht durchgeführt werden, während man mit dem Programm arbeitet.

```
--- MS-DOS Directory wird ausgedruckt ---
$type a.b
--- File a.b wird ausgedruckt ---
$^a^c
--- File c.b wird ausgedruckt ---
$!1:/c/cfile
--- File cfile.b wird ausgedruckt ---
$!d
```

Literatur

[1] *Kernighan, Ritchie:* The C Programming Language. Prentice Hall 1975.
[2] *Waite, Martin, Prata:* Unix Primer Plus. Howard W. Sums Inc. 1983/84.

Dieses in C geschriebene Programm emuliert Unix-C-Shell

```
#include <ctype.h>
#define AMAX 128
char h_array[AMAX][128];
char seg[128];
char seg1[128];
char seg2[128];
char dir[64];
int actuel = 0;
int current = 0;
int  wahl = 0;
char com_com[84] = "/c ";
main()
{
        char first = 0;
        int i;
        int id = 0;
        for(;;){/*display prompt */
                printf("$");
                gets(h_array[actuel]);
                eval_comm();
        }
}
eval_comm()
{
        com_com[0] = 0;
        if(!strcmp(h_array[actuel],"hi")) history();
        else if(!strcmp(h_array[actuel],"!!")) previous();
        else if(*h_array[actuel] == '!') prev_proc();
        else if(*h_array[actuel] == '^') edit_prev();
        else if(!strcmp(h_array[actuel],"pwd")) pwd_proc();
        else {
                strcat(com_com,h_array[actuel]);
                if(++actuel  == AMAX)actuel = 0;
                current++;
                fexecvp(com_com);
        }
}
fexecvp(s)
char *s;
{
        system(s);
}
prev_proc()
{  /* !n */
        register int i=0;
        int j;
                char c;
        while((c = h_array[actuel][++i]))
                if(!isdigit(c)){ /* USER MUST ENTER # OF ITEM */
                        printf("*** error %s\n",h_array[actuel]);
                        return;
                }
        i = atoi(h_array[actuel] + 1);
/* if i not in current-128 ... current error */
        if(i < current - AMAX || i >= current){
                printf("*** can't find %d\n",i);
                return;
        }
        if((j = actuel - (current-i)) >= 0)
                strcat(com_com,h_array[j]);
        else
                strcat(com_com,h_array[AMAX+1+j]);
        printf("%s",com_com);
        fexecvp(com_com);
}
```

Apple nutzt neue Grafik-Betriebsarten

Das Epson-Parallel-Interface „APL", das bei Apple-Besitzern weit verbreitet ist, bietet die Möglichkeit, zusammen mit grafikfähigen Epson-Druckern, auf bequeme Art einen Ausdruck der Apple-HGR-Seiten herzustellen. Hierzu wird nach Aktivierung der Schnittstelle (mit PR#n) durch CTRL-Q die Grafikdumproutine aufgerufen. Die APL-Firmware benutzt zum Aufruf des Grafikmodus die Steuersequenz ESC K bzw. ESC L für doppelte Druckdichte. Im ESC-K-Modus können jedoch nur maximal 480 Punkte pro Zeile gedruckt werden. Bei der Darstellungsart „Enlarged" (jeder Punkt wird in horizontaler und vertikaler Richtung verdoppelt) können daher nicht alle Bildschirmpunkte dargestellt werden. Bei doppelter Druckdichte (ESC L) wird dagegen das Bild gestaucht. Neuere Epson-Modelle sowie einige Epson-kompatible Drucker (z. B. Star, Citizen) bieten jedoch weitere Grafikmodi mit unterschiedlicher Druckdichte. Um sie zu benutzen, muß man die Grafikroutine der Interface-Firmeware etwas modifizieren. Die Steuersequenz ESC K bzw. ESC L muß durch die Sequenz ESC * n (n = Druckdichte 0...7, Default für Screendump: 5) ersetzt werden.

Hierzu verschiebt man nach Aktivierung der Karte durch PR#n den EPROM-Inhalt mit dem Monitor in den RAM-Bereich:

4000⟨C800.CFFFM

Da die neue Steuersequenz ein Byte länger ist als die alte, wird in einem freien Speicherbereich eine kurze Subroutine installiert, die das zusätzliche Zeichen ausgibt:

```
4056  : A9 05      LDA 05
40F0  : 20 E1 1C   JSR CCE1
        A9 2A      LDA #$2A
        20 E1 CC   JSR CCE1
        60         RTS
42C0  : 20 F0 C8   JSR C8F0
42FB  : 20 F0 C8   JSR C8F0
4476  : 20 F0 C8   JSR C8F0
44AC  : 20 F0 C8   JSR C8F0
```

Bei manchen Druckermodellen ist für eine verzerrungsfreie Wiedergabe noch der Zeilenvorschub von 8/12 auf 7/12 Zoll zu ändern:

```
4245  : A9 07      LDA #$07
```

Anschließend kann man die so modifizierte Routine in ein EPROM 2716 brennen, das dann gegen das Original auf der Interfaceplatine ausgetauscht wird.

Nach der Aktivierung der Karte kann jetzt durch „Poken" von Werten zwischen 0 und 7 in die Speicherstelle $478+n (n = Slot#) der Defaultwert der Punktdichte variiert werden. Um die Routine auch unter Turbo-Pascal nutzen zu können, fügt man dem Grafikpaket aus [1] noch folgende kurze Prozedur hinzu (Interface in Slot#1):

```
Procedure DUMP (r : Integer);
        Begin
            A_REG := $0;
            (* Init Interface *)
            Call6502 ($C100);
            (* HGR-Seite 2 *)
            MEM[$F779] := 2;
            (* 1:1 oder 1:2 *)
            If r = 2 Then MEM[$F779]
            := 66;
            (* CTRL-Q *)
            A_REG := $91;
            Call6502 ($C102);
        End;
```

Der Aufruf DUMP(1) liefert dann einen 1:1-Ausdruck, der Aufruf DUMP(2) einen Ausdruck im „Enlarged"-Modus. Durch entsprechendes Setzen des Grafik-Dump-Registers ($F779 unter CP/M) können auch die anderen Optionen der Routine, z. B. Invertieren, aufgerufen werden.

Gerhard Mehrke

Literatur

[1] *Schmidt, Hans-Jürgen:* Grafik für Apple-II-Turbo-Pascal. mc 1985, Heft 4, Seite 113...116.

```
}
previous()
{
        if(!current){
                printf("*** error <%s>\n",h_array[actuel]);
                return;
        }
        if(actuel)
                strcat(com_com,h_array[actuel -1]);
        else
                strcat(com_com,h_array[AMAX - 1]);
        printf("%s",com_com);
        fexecvp(com_com);
}
history()
{ /* print history */
        register int i,j;
        if(current < AMAX)
                for(i=0; i<current; i++)
                        printf("%d %s\n",i,h_array[i]);
        else{
                for(i = actuel+1,j = current - AMAX; i < AMAX; i++,j++)
                        printf("%d %s i=%d j=%d\n",j,h_array[i],i,j);
                for(i = 0,j = current - actuel ; i < actuel; i++,j++)
                        printf("%d %s i=%d j+%d\n",j,h_array[i],i,j);
        }
}
pwd_proc()
{
        int bx,cx,dx,di,si;
        si = dir;
        dx = 0;
        dos(0x47,bx,cx,dx,di,si);
        printf("%s\n",dir);
}
```

```
edit_prev()
{  /* !n */
        register int i = 1,k = 0;
        int j,l;
                char c;
        if(!current){
                printf("*** error <%s>\n",h_array[actuel]);
                return;
        }
        j = actuel ? actuel-1 : AMAX -1;
        while(( seg[k++] = h_array[actuel][i++]) != '^')
                if(i > 128){
                        printf("*** ^ not found error %s\n",h_array[actuel]);
                        return;
                } /* i = lgth of string */
        seg[k-1] = 0;
        l = 0;
        while(seg2[l++] = h_array[actuel][i++]);
        for(l = 0; l < strlen(h_array[j]) - k + 2; l++)
                if(!strncmp((h_array[j] + l),seg,k-1))goto nxt;
        printf("*** modifier failed <%s> <%s>\n",h_array[j],seg);
        return;
nxt:    strncpy(seg1,h_array[j] + l + k - 1,127 -l - k + 1);
        *(h_array[j] + l) = 0;
        strcat(h_array[j] + l,seg2);
        if(strlen(seg1) < 128 - l - k + 1){
                strcat(h_array[j] + l + k - 1,seg1);
                strcat(com_com,h_array[j]);
                printf("%s",com_com);
                fexecvp(com_com);
        }
        else {
                printf("string too long <%s>\n",seg1);
                return;
        }
```

Gerhard Rubel

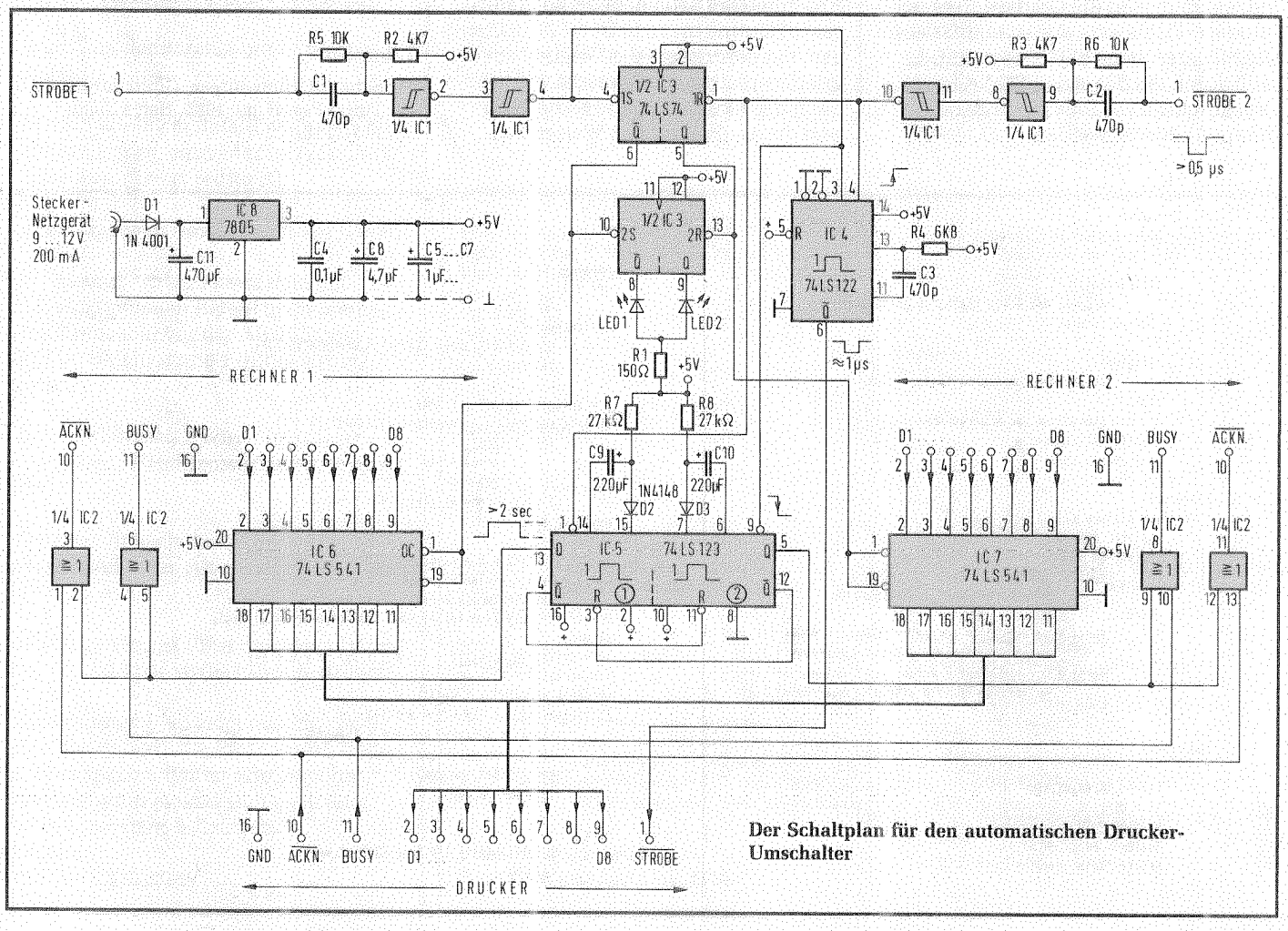
Automatischer Drucker-Umschalter

Zwei Computer an einen Drucker anschließen

Sollen zwei Computer an einen Drucker mit Centronics-Schnittstelle angeschlossen werden, so mußten früher entweder das Druckerkabel umgesteckt oder die Leitungen der beiden Computer zum Drucker mit einem vielpoligen Schalter umgeschaltet werden. mc-Leser, die den in diesem Beitrag vorgestellten automatischen Drucker-Umschalter nachbauen, brauchen sich nicht mehr um das Umstecken oder Umschalten zu kümmern.

Der automatische Drucker-Umschalter besteht aus drei Funktionsblöcken:
- Richtungsspeicher (IC 3)
- Verzögerungsschaltung für die beiden Strobe-Signale (IC 4)
- Datenbustreiber (IC 6, IC 7).

Um die beiden Rechnereingänge gegenseitig zu verriegeln, ist ein nachtriggerbares Monoflop (IC5: 74LS123) eingebaut. Es gibt je nach aktiviertem Rechner die Handshake-Signale Busy und Acknowledge frei. Die Schaltung kann erst auf den anderen Computer umschalten, wenn der derzeit aktive Computer ungefähr zwei Sekunden lang keine Daten gesendet hat. Der momentan aktive Computer wird durch eine der beiden grünen Leuchtdioden angezeigt. Mit den



Der Schaltplan für den automatischen Drucker-Umschalter

Tabelle: Die Stückliste

IC 1	74 LS14
IC 2	74LS32
IC 3	74LS74
IC 4	74LS122
IC 5	74LS123
IC 6, IC 7	74LS541
IC 8	7805
D 1	1N4001
D 2, D 3	1N4148
C 1, C 2, C 3	470 pF
C 4	0,1 µF
C 5...C 7	1 µF
C 8	4,7 µF
C 9, C 10	220 µF/10 V
C 11	470 µF/25 V
R 1	150 Ω
R 2, R 3	4,7 kΩ
R 4	6,8 kΩ
R 5, R 6	10 kΩ
R 7, R 8	27 kΩ
LED 1, LED 2	LED 5 mm grün
2 × 36polige Centronics-Einbaubuchsen 36poliger Centronics-Stecker	

R/C-Kombinationen R 5/C 1 und R 6/C 2 werden die Strobe-Eingänge entkoppelt. Ohne die Entkopplung würde die Schaltung nicht funktionieren, da bei einem ausgeschalteten Computer alle Signale der Centronics-Schnittstelle im L-Zustand sind. Der Drucker-Umschalter würde immer dem ausgeschalteten System den Zugriff zum Drucker zuteilen.

Wie funktioniert die Schaltung?

In unserer Beschreibung nehmen wir an, daß der Rechner 1 Daten sendet. Die Daten stehen am Eingang des Treibers (IC 6: 74LS541) an. Mit der negativen Flanke des Strobe-Signals am Eingang 1 wird ein Flipflop (IC 3: 74LS74) gesetzt und das zweite Monoflop von IC 5 getriggert. Der $\overline{Q}$-Ausgang (Pin 6) des Flipflops geht auf L – der Bustreiber (IC 6) wird aktiviert. Die Ausgänge des zweiten Bustreibers (IC 7) werden hochohmig, der Ausgang des zweiten Monoflops geht in den H-Zustand und die Signale Busy und Acknowledge des zweiten Centronics-Eingangs sperren die Datenübertragung des zweiten Rechners.
Da das erste Monoflop von IC 5 zurückgesetzt ist, geben die Signale Busy und Acknowledge die Datenübertragung des ersten Rechners frei. Durch die ansteigende Flanke des Strobe-Signals wird das Monoflop (IC 4) getriggert und der Strobe für den Drucker verzögert. Durch diese Verzögerung stehen die Daten sicher vor dem Strobe-Signal an. Bei jedem weiteren Strobe wird das zweite Monoflop von IC 5 nachgetriggert damit der andere Rechner gesperrt bleibt. Wenn ungefähr zwei Sekunden kein Signal kommt, fällt das zweite Monoflop ab und der zweite Rechner kann seine Daten zum Drucker senden. Der gleiche Vorgang erfolgt mit dem Datenbustreiber IC 7 und dem ersten Monoflop des IC 5. Als LED-Treiber wird das zweite Flipflop (IC 3) verwendet.
Zur Stromversorgung wird ein preiswertes Steckernetzteil (9...12 V/200 mA) verwendet. Die gesamte Schaltung (*Bild*) kann auf einer Lötpunktraster-Platine mit den Abmessungen 100 × 70 mm aufgebaut werden.

Symbolischer Disassembler für den Apple-II

Maschinenprogramme werden häufig als Hexdumps abgedruckt, weil der Sourcecode zu lang ist. Aber es gibt ja symbolische Disassembler: Sie machen aus dem Objektcode wieder Sourcecode. Ein solches Werkzeug stellen wir Ihnen hier vor.
Der Disassembler „OBJ2S" arbeitet mit dem im Monitor des Apple enthaltenen Disassembler zusammen. Mit „BRUN OBJ2S" gestartet, erscheint eine Installations- und Bedienungsmeldung; vom Monitor aus erfolgt der unmittelbare Aufruf mit Parameterübergabe. Mit „BSAVE" oder einem Basic-Programm zur Unterstützung ist die Sicherung des Ergebnisses als Binär- oder Textdatei möglich. Notwendige Parameter liefert das Programm selbständig zurück.
Die Verarbeitung des Objectcodes erfolgt in drei Schritten: Disassemblieren, Label-Verknüpfung und Label-Eliminierung. 1000 Zeilen Code werden pro Minute generiert, das entspricht etwa 2 KByte Objectcode.
Das Ausgabeformat (ASCII, Bit 7=1) ist dem Assembler „BigMac" angepaßt. Eine Fehlübersetzung von Zeropage-Indirekt-Befehlen durch den Apple-Monitor – leicht zu erkennen bei Vergleich der Labels mit Assembler-Label-Adressen – muß noch von Hand korrigiert werden. Die Labels erfassen absolute und relative Sprünge, Unterprogrammaufrufe sowie Datenbereiche; aus jedem Label ist die ursprüngliche Befehls- oder Daten-Adresse zu entnehmen.

Werner K. G. Münchheimer

Literatur

[1] *Baier, Erwin:* Disassembler. mc 1985, Heft 7, Seite 63...66.

[2] *Feichtinger, Herwig:* Prüfsummenprogramm für den Apple-II. mc 1984, Heft 6, Seite 64...65.


```
OBJ2S
A37632,L505
9300- A9 14 A0 93 8D F9 03 8C +=0405
9308- FA 03 A9 80 A0 94 20 3A +=03B4
9310- DB 4C D0 03 A9 C5 A0 94 +=049C
9318- 20 3A DB A9 25 A0 00 85 +=0328
9320- 18 84 17 A5 36 A4 37 85 +=02EE
9328- 1E 84 1F A9 31 A0 94 85 +=0354
9330- 36 84 37 A9 DD A0 94 20 +=03CB
9338- 3A DB A5 3D A6 3C 85 3B +=0399
9340- 86 3A 20 41 F9 20 8E FD +=03C5
9348- 20 8E FD A5 3D 20 71 94 +=03B2
9350- 85 CE A5 3F 20 71 94 85 +=03E1
9358- CF A9 00 20 ED FD A5 3B +=0462
9360- A6 3A 20 41 F9 A2 01 20 +=02FD
9368- 4A F9 20 8C F8 A2 03 20 +=03AC
9370- EA F8 20 8E FD 20 53 F9 +=04F9
9378- 85 3A 84 3B C5 3E 98 E5 +=03FE
9380- 3F 90 D6 A9 25 A0 1C 85 +=03B4
9388- 3D 85 41 85 43 84 3C 84 +=030F
9390- 40 84 42 A0 00 B1 3C C9 +=035C
9398- A4 D0 4A C8 A5 CF D1 3C +=0507
93A0- 90 43 B1 3C C5 CE 90 3D +=0420
93A8- A0 03 B1 3C C9 B0 90 35 +=03CE
93B0- A9 25 A0 1C 85 3F 84 3E +=0310
93B8- A0 00 B1 3E F0 04 C9 C7 +=0413
93C0- D0 13 A0 04 B1 3C D1 3E +=0383
93C8- D0 0B 88 D0 F7 A9 C7 91 +=052B
93D0- 3C 91 3E D0 10 A5 3E C5 +=0393
93D8- 17 A5 3F E5 18 E6 3E D0 +=03EC
93E0- 02 E6 3F 90 D3 A5 3C C5 +=0430
93E8- 17 A5 3D E5 18 E6 3C D0 +=03E8
93F0- 02 E6 3D 90 9E A0 00 B1 +=03A4
93F8- 40 91 42 D0 04 A0 04 D0 +=035B
9400- 06 E6 42 D0 02 E6 43 E6 +=040F
9408- 40 D0 02 E6 41 88 10 F7 +=03C8
9410- A5 40 C5 17 A5 41 E5 18 +=03A4
9418- 90 DB A5 1E A6 1F 85 36 +=03AE
9420- 86 37 A5 43 E9 25 A6 42 +=039B
9428- 20 41 F9 20 8E FD 4C D0 +=0421
9430- 03 84 46 A0 00 C9 BF D0 +=03C5
9438- 23 C5 19 F0 2F A2 03 BD +=0382
9440- F5 94 20 64 94 CA 10 F7 +=0472
9448- B1 3A 48 20 71 94 20 64 +=02DC
9450- 94 68 20 75 94 20 64 94 +=033D
9458- A9 BF D0 10 C9 A0 D0 04 +=0485
9460- C5 19 F0 08 91 17 E6 17 +=037B
9468- D0 02 E6 18 A4 46 85 19 +=0358
9470- 60 4A 4A 4A 4A 29 0F 09 +=01C9
9478- B0 C9 BA 90 02 69 06 60 +=0394
9480- 8D 8D CF C2 CA B2 D3 BA +=05B4
9488- A0 CD C1 C3 A0 D3 CF D5 +=0608
9490- D2 C3 C5 A0 C7 C5 CE C5 +=0619
9498- D2 C1 D4 CF D2 A0 C2 D9 +=0643
94A0- A0 D7 CD 8D 8D AA B9 B3 +=0574
94A8- B0 B0 C7 A0 C3 CF CC C4 +=05E9
94B0- 8D 8D AA D3 D4 C1 D2 D4 +=05D2
94B8- AE C5 CE C4 A0 C3 D4 D2 +=060E
94C0- CC AD D9 8D 00 8D C2 D3 +=0501
94C8- C1 D6 C5 A0 C6 C9 CC C5 +=061C
94D0- AE D3 AC C1 A4 B2 B5 B0 +=05A9
94D8- B0 AC CC A4 00 AA A0 CF +=04E5
94E0- C2 CA B2 D3 A0 CF D5 D4 +=0629
94E8- D0 D5 D4 8D AA 8D A0 CF +=05AC
94F0- D2 C7 A0 A4 00 A0 D8 C5 +=051A
94F8- C8 +=00C8
```

Mit dem im Apple-Monitor enthaltenen Disassembler arbeitet das Programm zusammen

Jürgen Plate

Software für Grafiktablett

Grafik-Software – preiswert serviert

Unter dem Titel „Grafik preiswert serviert" haben wir vor etwa einem Jahr ein Grafiktablett vorgestellt, das sich nicht nur durch seine Leistung, sondern auch durch einen besonders niedrigen Preis auszeichnete. Als Folge des Artikels kam der Hersteller, CRP-Koruk in Konstanz, kaum mit dem Liefern nach. Seit einiger Zeit wird das Grafiktablett mit unterstützender Software geliefert, die wir uns einmal näher angesehen haben.

Nachdem ich das Tablett selbst besitze, war ich natürlich recht gespannt, was der Kunde eines 1500 DM-Gerätes als zusätzliches Zuckerstück dazu bekommt. Auf der ersten von drei Disketten befinden sich ein Demo-Programm mit ganz einfachen Zeichenfunktionen sowie ein paar Beispiele für die Ansteuerung in Turbo-Pascal und GW-Basic für den IBM-PC. Auf zwei weiteren Disketten befindet sich das Programm TAKAVOR, ein Malprogramm, das tatsächlich einen guten Einstieg in die Bildverarbeitung erlaubt und das ich im folgenden beschreiben werde. Das Programm kann für 380 DM erworben werden. Herr Heiduck von CRP hat jedoch das Angebot gemacht, daß die Kunden „der ersten Stunde", die sich unter Vorlage einer Rechnungskopie des Grafiktabletts auf diesen Testbericht beziehen, einen Sonderpreis erhalten.
TAKAVOR wird mit einem englischen Handbuch geliefert und ist (jedenfalls aus meiner Sicht) sehr leicht zu erlernen. Das Programm arbeitet mit allen gebräuchlichen Grafikkarten und Matrixdruckern zusammen (auch Farbdruckern), ein Treiber für den HP Laserjet ist ebenfalls erhältlich. Es stellt sich dem Benutzer mit dreigeteiltem Bildschirm dar (*Bild 1*). Den größten Teil des Bildschirms beansprucht das Zeichenfeld, das Teil eines sogenannten „virtuellen Schirms" sein kann, dessen Größe vom Typ des Druckers bestimmt wird. Alle Zeichen- und Löschoperationen finden nur im gerade sichtbaren Zeichenfeld statt. Einen anderen Teil des virtuellen Bildschirms kann man mit dem Feld ganz rechts unten wählen und dann für eine größere Zeichnung anschließen.
Links neben dem Zeichenfeld sind die Piktogramme für die einzelnen Zeichenfunktionen. Piktogramme mit einem kleinen Quadrat führen auf Wunsch in ein Untermenü, das oben im Zeichenfeld „aufgeklappt" wird. Ganz links unten sieht man die gerade aktiven Funktionen und die aktuelle Stiftfarbe. Die Felder unter dem Zeichenbereich gestatten die Wahl der Stiftbreite, der Zeichenfarbe und des Musters für die Füllfunktion.
Kreise, Ellipsen, Rechtecke, Schrift und Linien lassen sich durch Antippen des entsprechenden Piktogramms auswählen und dann auf die entsprechende Größe ziehen. Es gibt Funktionen zum Freihandzeichnen, zum Ziehen von Linien im Gummifadenmodus und sogar zur Spline-Interpolation durch eine Reihe von Punkten. Für genaues Arbeiten kann ein Raster unterlegt werden. Für besondere Anwendungen läßt sich ein „Sprühdoseneffekt" einsetzen. Flächenfüllen ist ebenso möglich wie das Verschieben, Löschen oder Kopieren von Bildausschnitten. Besonders hat mir die Funktion zum Spiegeln, Austauschen und Drehen von Bildausschnitten gefallen. Natürlich gibt es einen Radiergummi zum Löschen von unerwünschten Bildteilen. Mit der Feinkorrekturfunktion lassen sich Bildausschnitte vergrößert darstellen und Bildpunkt für Bildpunkt korrigieren.
Mit TAKAVOR hat man auch eine ganze Reihe von Schriftarten zur Verfügung, mit deren Hilfe sich Einladungen, Grußkarten oder Gutscheine wie in *Bild 2* erzeugen lassen. Schrifthöhe und -breite lassen sich immer passend einstellen. Sie können auch Texte aus Textdateien direkt einlesen, sofern sie das Bildformat nicht sprengen.
Zusammen mit dem Hauptprogramm kommen noch zwei Hilfsprogramme, die den Wert von TAKAVOR potenzieren. Das eine ist ein residentes Programm namens GRAB, das es erlaubt, Screen-Dumps im passenden Format auf Diskette zu speichern. Man könnte so

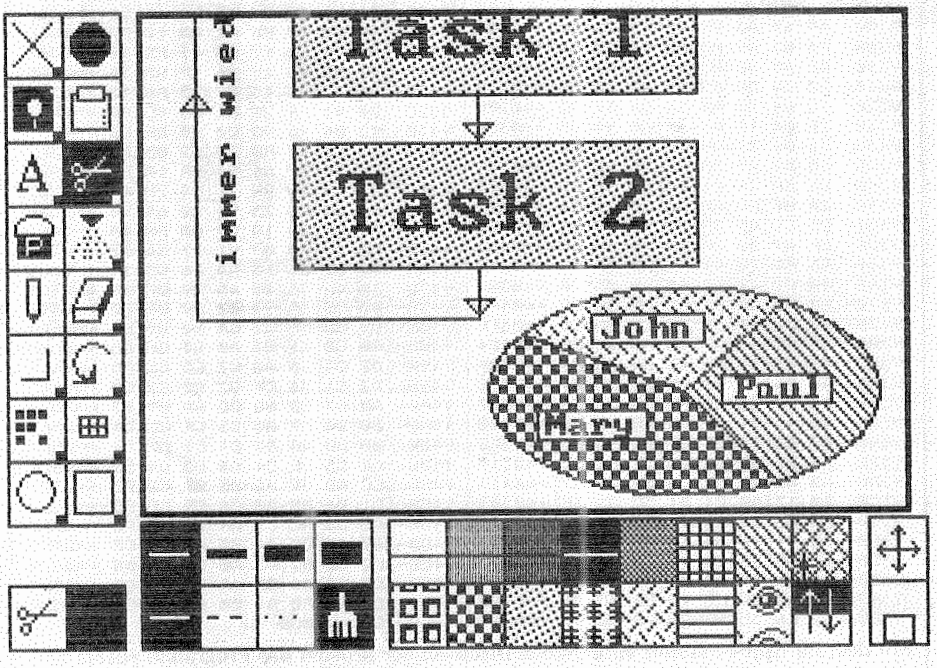


Bild 1. Das Programm Takavor ist durch Menütechnik einfach zu bedienen

Bild 2. Die Gestaltung von Grußkarten wird durch zahlreiche Schriftarten vereinfacht

die durch ein beliebiges Programm erzeugten Grafiken weiterbearbeiten. Das zweite Programm TSHOW erlaubt den Aufbau einer Art Diashow. Grafiken lassen sich entweder durch Zeitablauf oder durch Tastendruck gesteuert fortlaufend ausgeben – z. B. für Demonstrationen. TAKAVOR ist sicher nicht „das" Superprogramm zum Lösen aller Grafikprobleme – das soll es aber ganz bewußt auch nicht sein. In dem Bereich, den es abdecken soll, leistet es gute Arbeit und ich werde es sicher gerne und oft verwenden. Was ich bemerkenswert finde, ist die Philosophie des Herstellers, der nicht nur das Grafiktablett verkauft (nach dem Motto: „läuft mit ...."), sondern den Kunden mit Software unterstützt (es gibt neben TAKAVOR eine ganze Reihe von Programmen und Bibliotheken bei CRP).

Siemens PC-D mit hoher Auflösung

In mc 11/86 wurde im Beitrag „Hohe Auflösung mit dem M-24" geschildert, wie unter Turbo-Pascal der hochauflösende Modus dieses PCs angesprochen werden kann. Das Assembler-Programm braucht nur an einigen Stellen geändert zu werden und schon kann Turbo-Pascal den hochauflösenden Modus der Computer Siemens PC-D und Tandy-2000 aufrufen.

Die Programmzeilen in Turbo-Pascal:

```
Procedure Hires; external 'hires.bin';
Procedure Plot (X,Y:Integer); external
hires [3];
Procedure Radi (X,Y:Integer); external
hires [6];
```

Manfred Weis

```
0000                    ;*    HIRESO  Routinen zur grafischen      *
                        ;*            Ansteuerung aus TURBO        *
                        ;*                                         *
                        ;*    HIRESO : Initialisierung             *
                        ;*    PLOT   : Punkt setzen                *
                        ;*    RADI   : Punkt loeschen    OKT 1986  *
                        ;*                                         *
                        ;*******************************************
                        ;
0000                    code    segment para public 'code'
                                assume  cs:code
0000  EB 07 90                  jmp     hireso  ;VKT Adresse
0003  EB 0A 90                  jmp     plot    ;Punkt setzen
0006  EB 4E 90                  jmp     radi    ;Punkt loeschen
0009  B8 E000           hireso: mov     ax,0E000h
000C  8E C0                     mov     es,ax
000E  C3                        ret
000F  55                plot:   push    bp
0010  8B EC                     mov     bp,sp   ;Stackzugriff erlaubt
0012  8B 76 06                  mov     si,[bp+6]       ;X-Koord
0015  83 FE 00                  cmp     si,0            ;X<0 ?
0018  7C 36                     jl      endpl
001A  81 FE 027F                cmp     si,639          ;X>639 ?
001E  7F 30                     jg      endpl
0020  8B 7E 04                  mov     di,[bp+4]       ;Y-Koord
0023  83 FF 00                  cmp     di,0            ;Y<0 ?
0026  7C 28                     jl      endpl
0028  81 FF 018F                cmp     di,399          ;Y>399 ?
002C  7F 22                     jg      endpl   ;Zeilenberechnung
002E  8B DF                     mov     bx,di   ;BX:=Y
0030  B1 06                     mov     cl,6
0032  D3 E3                     shl     bx,cl   ;BX:=Y*64
0034  8B C7                     mov     ax,di
0036  B1 04                     mov     cl,4
0038  D3 E0                     shl     ax,cl   ;AX:=Y*16
003A  03 D8                     add     bx,ax   ;BX:=Y*80
003C  8B C6                     mov     ax,si   ;AX:=X
003E  B1 03                     mov     cl,3
0040  D3 E8                     shr     ax,cl   ;AX:=X div 8
0042  03 D8                     add     bx,ax   ;BX:=Byte des Punktes
0044  B9 0007                   mov     cx,7
0047  23 CE                     and     cx,si   ;CX:=X mod 8
0049  B0 80                     mov     al,80h  ;7-(x mod 8)te Bit
004B  D2 E8                     shr     al,cl   ;selektieren
004D  26: 08 07                 or      es:[bx],al
0050  8B E5              endpl:  mov     sp,bp
0052  5D                        pop     bp
0053  C2 0004                   ret     4
0056  55                radi    push    bp
0057  8B EC                     mov     bp,sp   ;Stackzugriff erlaubt
0059  8B 76 06                  mov     si,[bp+6]       ;X-Koord
005C  83 FE 00                  cmp     si,0            ;X<0 ?
005F  7C 38                     jl      stopp
0061  81 FE 027F                cmp     si,639          ;X>639 ?
0065  7F 32                     jg      stopp
0067  8B 7E 04                  mov     di,[bp+4]       ;Y-Koord
006A  83 FF 00                  cmp     di,0            ;Y<0 ?
006D  7C 2A                     jl      stopp
006F  81 FF 018F                cmp     di,399          ;Y>399 ?
0073  7F 24                     jg      stopp   ;Zeilenberechnung
0075  8B DF                     mov     bx,di   ;BX:=Y
0077  B1 06                     mov     cl,6
0079  D3 E3                     shl     bx,cl   ;BX:=Y*64
007B  8B C7                     mov     ax,di
007D  B1 04                     mov     cl,4
007F  D3 E0                     shl     ax,cl   ;AX:=Y*16
0081  03 D8                     add     bx,ax   ;BX:=Y*80
0083  8B C6                     mov     ax,si   ;AX:=X
0085  B1 03                     mov     cl,3
0087  D3 E8                     shr     ax,cl   ;AX:=X div 8
0089  03 D8                     add     bx,ax   ;BX:=Byte des Punktes
008B  B9 0007                   mov     cx,7
008E  23 CE                     and     cx,si   ;CX:=X mod 8
0090  B0 80                     mov     al,80h  ;7-(x mod 8)te Bit
0092  D2 E8                     shr     al,cl   ;selektieren
0094  F6 D0                     not     al
0096  26: 20 07                 and     es:[bx],al
0099  8B E5              stopp:  mov     sp,bp
009B  5D                        pop     bp
009C  C2 0004                   ret     4
009F                    code    ends
                                end
```

Mit diesem Assembler-Programm wird die hochauflösende Grafik angesprochen

Reiner Schönrock

Boeing-Calc

Dreidimensionale Tabellenkalkulation

Wer mit Programmen wie Lotus 1-2-3 oder Symphony an Grenzen stieß, weil der Speicherplatz nicht ausreichte oder die einzelnen Spreadsheets zu klein waren, kann jetzt fast ohne Beschränkung agieren. Mit Boeing-Calc kommt ein Tabellenkalkulationsprogramm für PCs auf den Markt, das mehrere Spreadsheets miteinander verknüpft.

Der Weltkonzern Boeing mit Sitz in Seattle ist jedem als ein Unternehmen, das sich hauptsächlich mit der Produktion und Entwicklung von Düsenflugzeugen beschäftigt, bekannt. Im Jahre 1970 wurde die Boeing Computer Services Company in Bellevue, Washington, gegründet. Heute arbeiten über 12500 Mitarbeiter in diesem Geschäftsbereich, der zum Beispiel in der Forschung und Entwicklung neuer Luftfahrzeuge involviert ist. Mehrere Projekte der Künstlichen Intelligenz, ein Telekommunikationsnetz für die NASA und ein Beitrag zur automatisierten Fabrik in Zusammenarbeit mit General Motors sind weitere Felder, auf denen sich Boeing mit seiner Computer Division betätigt.

Aus den Erfahrungen, die bei Boeing durch den Einsatz von Tabellenkalkulationsprogrammen gemacht wurden, entstand das Programm Boeing-Calc, das für etwa 1250 DM jetzt auch in Deutschland zu haben ist. An einer rein deutschen Version des Programms wird noch gearbeitet; sie soll aber bald fertig sein. Der große Vorteil, den Boeing-Calc gegenüber anderen Programmen hat, ist, daß mehrere Spreadsheets untereinander sehr einfach verknüpft werden können. Dieser Sachverhalt ist für die Namensgebung „dreidimensionales Spreadsheet" verantwortlich (*Bild 1*). Mit Grafik, die das Programm nicht darstellen kann – auch nicht bei zweidimensionalen Spreadsheets – hat das nichts zu tun.

Boeing-Calc läuft auf jedem IBM PC/XT/AT oder Kompatiblen mit mindestens 384 KByte Hauptspeicher. Wer den Hintergrund, den Rahmen oder einzelne Bereiche eines Arbeitsblattes farbig darstellen möchte, kann das mit Boeing-Calc einfach tun. Bei mehreren Spreadsheets, die übereinanderlappend auf dem Bildschirm dargestellt werden können, kann jeder Seite eine eigene Farbe zugewiesen werden. Der zum Einsatz kommende Computer muß eine Festplatte eingebaut haben, da alleine das EXE-File über 460 KByte groß ist. Wir haben das Programm auf einem AT von MCI, dem AT4SLC mit 512 KByte RAM und einer 20 MByte Festplatte, getestet. Ohne jedwede Installation konnte sofort mit dem Programm gearbeitet werden. Die integrierten Hilfefunktionen geben auf Wunsch sehr ausführliche Hinweise zur richtigen Benutzung. Auf den Programmdisketten ist ein Tutorial enthalten, das bei den ersten Schritten durch die Spreadsheets hilft.

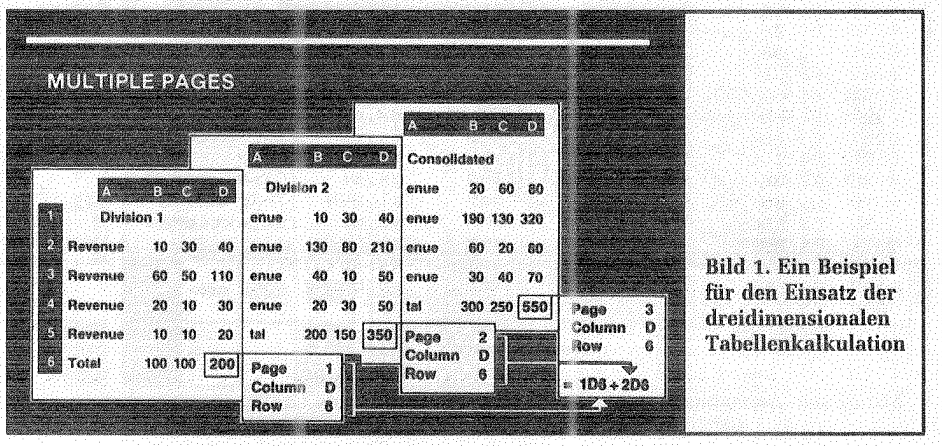


Bild 1. Ein Beispiel für den Einsatz der dreidimensionalen Tabellenkalkulation

Bis zu 32 MByte

Anwender von Lotus 1-2-3 oder Symphony konnten ihre Tabellenkalkulationen bislang nur im Rahmen des zur Verfügung stehenden Speicherplatzes durchführen. Maximal sind mit diesen Programmen 4 MByte adressierbar. Ist ein Spreadsheet größer, muß es in mehrere kleine geteilt werden. Boeing-Calc zeigt hier neue Dimensionen auf, denn es schreibt alle Daten, die im Hauptspeicher keinen Platz mehr finden, direkt auf die Festplatte. So kann ein Spreadsheet bis zu 32 MByte umfassen. Die maximale Größe eines Arbeitsblattes beträgt bei Boeing-Calc 16 000 Zeilen zu 16 000 Spalten. Von diesen Riesen-Spreadsheets können 16 000 in einer Datei angelegt werden. Das sollte auch für die umfangreichsten Kalkulationen reichen.

Die Befehlsstruktur von Boeing-Calc ist der von Lotus 1-2-3 ähnlich. Wer sich mit diesem Programm auskennt, wird keine Probleme bei der Bedienung bekommen. Auch bereits vorhandene Daten im 1-2-3- oder Symphony-Format können in Boeing-Calc übernommen werden. Lediglich bei der Bezeichnung der Arbeitsblätter muß man nun noch die Seite mit angeben. Von der ersten Seite, der ersten Zeile und Spalte einer Datei (1A1) bis zur letzten adressierbaren Zelle (16000WQJ16000) sind alle Zellen miteinander verknüpfbar. Auf den Anwender kommt mit Boeing-Calc ein wahrhaft riesiges Programm zu, das ein Umdenken bei der Bedienung erfordert. Spreadsheets waren bislang relativ klein und einfach zu überblicken. Mit der dritten Dimension von Boeing-Calc wird zur Nutzung der höheren Leistung auch mehr Disziplin beim Einsatz gefordert.

Wer braucht das?

Bei der Beschreibung der technischen Leistungsmerkmale wird sich der eine oder andere fragen, wofür man denn die Dritte Dimension eigentlich braucht. Dazu einige Anregungen: Jeder größere Betrieb muß für die einzelnen Bereiche Kostenrechnungen durchführen. Mit Boeing-Calc wird nun für jede Abteilung ein Arbeitsblatt eingerichtet und ausgefüllt. Die einzelnen Dateien können dann – auf einem PC – zusammengeführt und

ausgewertet werden. Da sich Berechnungen bei diesem Programm über mehrere einzelne Seiten erstrecken können, ist es kein Problem mehr, Gesamtergebnisse zu erhalten. Für die Ergebnisse eines Monats kann dann zum Beispiel wieder eine Seite angelegt werden. Am Ende eines Geschäftsjahres werden die einzelnen Monatsdaten dann zu einer Gesamtbilanz zusammengeführt.
Auch in der Forschung ist der Einsatz von Boeing-Calc sinnvoll. Jeder einzelnen Messreihe wird ein Spreadsheet zur Verfügung gestellt. Beim Anschluß der Versuche werden die einzelnen Ergebnisse – auch aus unterschiedlichen Abteilungen – zusammengetragen und ausgewertet.

Auch im Netz

Von Boeing-Calc gibt es für 2500 DM auch eine netzwerkfähige Version. Bis zu 32 Benutzer können gleichzeitig an ein und demselben Spreadsheet arbeiten. Dabei kann jeder Anwender in einem beliebigen Bereich des Spreadsheets arbeiten, ohne andere Bereiche zu beeinflussen. Die netzwerkfähige Version benötigt MS-DOS 3.0 und IBM Token Ring- oder PC/NET-Software.
Die Neuberechnung eines Arbeitsblattes kann in Boeing-Calc auf bestimmte Bereiche begrenzt werden. In der Ausführungsgeschwindigkeit ist es mit Lotus 1-2-3 vergleichbar. Der Co-Prozessor 8087 wird unterstützt.

Datensicherung bei Festplattenlaufwerken

Anwender, die in ihrem Rechner eine Festplatte einsetzen, werden mit der Frage konfrontiert, wie die Dateien der Festplatte gesichert werden sollen. Das Backup auf Disketten ist eine äußerst mühselige und zeitraubende Tätigkeit, da 92 Disketten im 360-KByte-Format gebraucht werden, um alle Dateien einer vollbelegten 32-MByte-Festplatte zu sichern. Auch bei Disketten im 1,2-MByte-Format wären immer noch 27 Disketten notwendig. Eine Lösung dieses Backup-Problems bieten Streamer.
Ein Streamer ist ein Bandlaufwerk, das die Daten des Plattenlaufwerks auf einem Magnetband speichert. Streamer sind inzwischen so kompakt gebaut, daß sie nicht mehr Platz beanspruchen als ein 5¼"-Laufwerk in halber Bauhöhe. Der japanische Hersteller Teac, vertreten durch nbn-Elektronik, Herrsching, hat die Streamer-Serie MT-2ST-Serie auf den Markt gebracht.
Als Speichermedium wird eine Kassette verwendet, eine Weiterentwicklung der schon 1968 vorgestellten digitalen Datenkassette. Wurden bisher Bandlaufwerke im wesentlichen zur Datensicherung eingesetzt, so zeichnen sich heute weitere Anwendungen ab. Software-Pakete werden immer komplexer und umfangreicher. Das Installieren eines Programms auf Festplatte macht den Anwender häufig zu einem Diskjockey, da selbst Programme wie Wordstar-Extra auf sechs Disketten geliefert werden.
Bei digitalen Datenkassetten mit ihrem gegenüber Disketten wesentlich höheren Speichervermögen braucht nur die Kassette in den Streamer eingelegt und das Übertragungsprogramm gestartet werden.

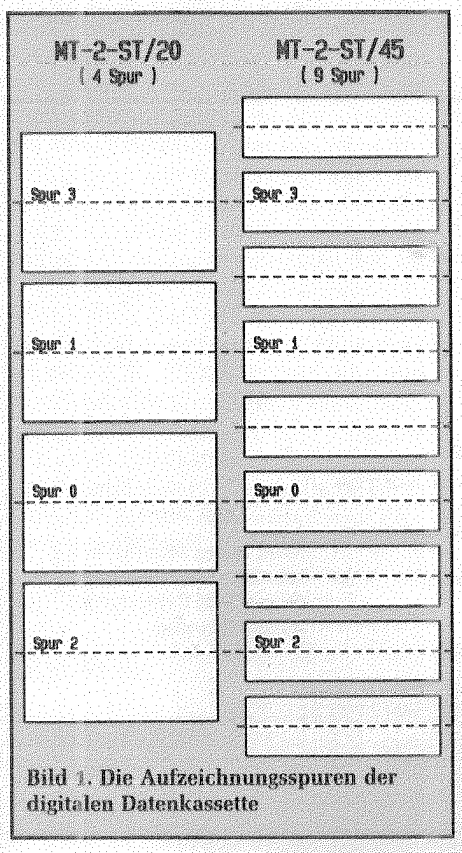


Bild 1. Die Aufzeichnungsspuren der digitalen Datenkassette

Die Teac-Streamer der Serie MT 2ST sichern 20 MByte im 4-Spur-Serpentinenverfahren in vier Minuten und 60 MByte auf neun Spuren in 10 Minuten (*Bild 1*). Eine Einsteckkarte, deren E/A-Adresse im Bereich 000...3FCH einstellbar ist, paßt den Streamer an den PC-Bus an (*Bild 2*). Daten werden mit 86,3 KByte/s übertragen. Der PC sollte mit 256 KByte RAM ausgerüstet sein.

Die mitgelieferte Software läuft ab PC-DOS 2.01. Das Platten-Backup kann dateiweise oder als direktes Spiegelbild der Platte durchgeführt werden. Die Übertragung in ganzen Dateien dauert länger, da erst die gesamten Cluster einer Datei gesucht werden müssen. Wird der Inhalt einer Festplatte als direktes Spiegelbild auf das Band übertragen, d. h., der Festplatteninhalt wird Spur für Spur auf Band gesichert, so versagt diese Backup-Methode beim Überspielen des Bandes auf eine andere Festplatte. Hat diese Festplatte andere defekte Spuren als die ursprüngliche Festplatte, so verschwinden alle Daten, die auf diese Spur übertragen werden, in einem „schwarzen Loch". Zum Archivieren von Daten sollte das Backup im Datei-Modus durchgeführt werden.

Nach Unterlagen von nbn-Elektronik

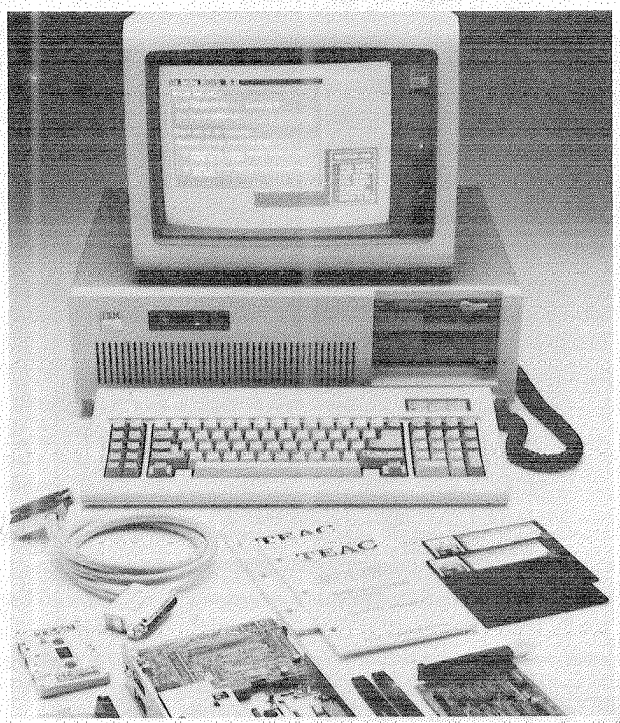


Bild 2. Einfaches Backup der Festplatte mit den Streamern der MT-2-Familie

Reiner Schönrock

Omikron-Basic

Interpreter im Modul für den Atari ST

Der Basic-Interpreter von Omikron rechnet auf 19 Stellen genau, enthält Befehle zur Multitask-Programmierung und ist absturzsicher in einem ROM-Modul untergebracht. Für rechenintensive und umfangreiche Aufgaben ist dieses Basic prädestiniert.

Der Omikron Basic-Interpreter wird in einem Modul für den Atari ST angeboten. Für 229 DM bekommt man die drei EPROMs vom Typ 27265 auf einer Steckplatine, ein umfangreiches Handbuch, eine Klebefolie für die Funktionstasten und eine Diskette mit diversen Funktions-Bibliotheken und Programmen. Auf der Diskette ist auch der frei kopierbare Runtime-Interpreter enthalten. Für den Programmierer ergibt sich bei diesem Basic der Nachteil, daß am Interpreter durch Patches, wie es sie für das GfA-Basic gibt, nichts geändert werden kann.
In zwei EPROMs ist die ROM-Floppy und der Basic-Interpreter (liegt von 0290H bis FFB7H) untergebracht. Das dritte EPROM enthält die Initialisierungsdaten für den RAM-Bereich des Interpreters (von 0000 bis 2C89H) und den Full-Screen-Editor (von 3000H bis 7D5EH).

Die Inbetriebnahme

Das EPROM-Modul wird in den EPROM-Schacht des (ausgeschalteten) Atari ST gesteckt. Das Betriebssystem des Atari ST installiert das Modul als „c ROM-Modul", sofern nicht die Omikron-Diskette zum Booten benutzt wird. Wer jetzt das neue Icon erwartungsvoll anklickt, wird allerdings enttäuscht, denn nach 14maligem Aufruf des Moduls „c" würde das System abstürzen. Das Modul muß mit einem anderen Kennzeichen installiert werden. Verwendet man zum Booten die Omikron-Diskette, wird automatisch der Kennbuchstabe „O" für das Modul eingesetzt. Außerdem enthält die Datei „Desktop.Inf" die Information, daß beim Anklikken von Dateien mit der Extension ".bas" automatisch das Omikron-Basic gestartet und das angewählte Programm geladen wird.
Die Programmierer bei Omikron sind auf Nummer Sicher gegangen und verhindern einen Aufruf des Interpreters von Modul „c". Sollte man es trotzdem versuchen, erhält man einen entsprechenden Hinweis.
Je nachdem, wieviele Desk-Accessories in den Speicher des Atari ST geladen werden, steht freier Speicherplatz zur Verfügung. Omikron-Basic zeigt nach dem Aufruf des Interpreters durch Starten des „OM-BASIC.PRG" im Modul „O" (70 Bytes groß) den verfügbaren Speicherplatz an. Maximal sind 865496 Bytes frei. Wer mit einer Festplatte arbeitet, kann über maximal 854174 Bytes RAM verfügen. Diese Angaben beziehen sich auf einen Atari ST mit 1 MByte RAM und dem Betriebssystem im ROM. Das Modul kann auch mit dem aktuellen Disketten-TOS eingesetzt werden.

Basic ist nicht gleich Basic

Omikron hat sich bei der Implementierung des Interpreters weitgehend an den MBasic-Dialekt von Microsoft angelehnt. Die größte Gemeinsamkeit – und damit ein großer Unterschied zum GfA-Basic – besteht darin, daß Omikron-Ba-

```
12 FOR X=0 TO W
15 FOR Y=0 TO H
19 FOR Nr=1 TO 4
33     FOR I=0 TO 7:Message(I)= CVI( MID$(X$,I%2+1,2)
): NEXT
41         FOR Nr=1 TO 4
281     FOR I=1 TO 10
462     FOR I=0 TO 62 STEP 2: WPOKE Graf_Mouse+10+I,
 WPEEK(T+I): NEXT
547     FOR I=1 TO LEN(Text$): POKE X+I-1, ASC( MID$
(Text$,I)): NEXT : POKE X+I-1,0
653     FOR I=0 TO LEN(Text$)-1:Intin%(I)= ASC( MID$
(Text$,I+1,1)): NEXT
713     FOR I=1 TO LEN(Text$):Intin%(I+1)= ASC( MID$
(Text$,I)): NEXT :Intin%(I+1)=0
819     FOR I=1 TO Ctrl%(4):Text$=Text$+ CHR$(Intout
%(I-1)): NEXT
826     FOR I=1 TO Ctrl%(4):Text$=Text$+ CHR$(Intout
%(I-1)): NEXT
910     LOCAL I: FOR I=0 TO 36:Intin%(I)= CVI( MID$(
Text$,I%2+1)): NEXT
940     FOR I=1 TO 32:Text$=Text$+ CHR$(Intout%(I)):
NEXT
1032        FOR I=1 TO LEN(Text$):Intin%(I+1)= ASC( M
ID$(Text$,I,1)): NEXT :Intin%(I+1)=0
1046    FOR I=0 TO 124:Text$=Text$+ CHR$(Intout%(I))
: NEXT
1087        FOR I=1 TO LEN(Text$):Intin%(I-1)= ASC( MI
D$(Text$,I,1)): NEXT :Intin%(I+1)=0
1095     FOR I=1 TO LEN(Text$):Intin%(I-1)= ASC( MID
$(Text$,I)): NEXT
1166      Intin%(0)=X'RESERVED FOR FUTURE USE

END OF LISTING. PRESS ANY KEY.
```

```
   IF Message(0)=Mn_Selected THEN
      IF Message(4)=Message THEN
         Form_Alert(1,"[1][Dies ist ein Beispielpro
gramm!|für GEM unter OMIKRON-BASIC][ OK ]",0)
         Menu_Tnormal(Desk,1)
      ENDIF
      IF Message(4)=Quit THEN
         Graf_Mouse(2)
         FOR Nr=1 TO 4
            IF Wind_Open(Nr) THEN
               Wind_Close(Wind_Handle(Nr))
            ENDIF
            Wind_Delete(Wind_Handle(Nr))
         NEXT Nr
         Menu_Tnormal(File,1)
         Appl_Exit
         END
      ENDIF
      IF Message(4)=Openw THEN
         Nr=1
         WHILE Wind_Open(Nr):Nr=Nr+1 WEND
         IF Nr>4 THEN
            Form_Alert(1,"[3][Zu viele Windows][ Ab
bruch ]",0)
         ELSE
            Window=Window_Handle(Nr):X=X(Nr):Y=Y(Nr
):W=W(Nr):H=H(Nr)
            Form_Dial(1,X,Y,W,H)
            Wind_Open(Window,X(Nr),Y(Nr),W(Nr),H(Nr
)):Wind_Open(Nr)=True
         ENDIF
         Menu_Tnormal(Twindow,1)
      ENDIF
   ENDIF
   IF Message(0)=Wm_Topped THEN
      Nr=1:Window=Message(3)
      WHILE Nr<5 AND Window_Handle(Nr)<>Window:Nr=N
r+1 WEND
      IF Nr<5 THEN
         IF Wind_Open(Nr) THEN
            Wind_Set(Window)
         ENDIF
      ENDIF
```

Bild 1. Der Monitor kann zur besseren Übersicht in bis zu drei Bereiche aufgeteilt werden

sic mit Zeilennummern arbeitet und in eine Zeile mehrere Befehle geschrieben werden dürfen. MBasic-Programme – aber auch Basic-Programme aus anderen Dialekten – können mit dem Omikron-Basic verarbeitet werden. Der Interpreter stellt Zeilen, die er nicht versteht, invers dar. Da die meisten Basic-Dialekte mit mehreren Befehlen in einer Zeile arbeiten, ist die Übernahme von Programmen in das Omikron-Basic sehr einfach, weil kaum Änderungen nötig sind.
Um die Übernahme von anderen Basic-Programmen zu testen, haben wir das integrierte Datenbank-Programm (IDB) aus der mc 5/85 als ASCII-Datei auf den Atari ST überspielt und mit Omikron-Basic geladen. Der Interpreter monierte lediglich COLOR-, LOCATE- und ERASE- Anweisungen.
Bei der INKEY$-Funktion zeigte das Omikron-Basic bei bestimmten Tasten ein ungewöhnliches Verhalten. Diese Funktion liefert beim Atari ST pro Tastendruck einen vier Byte langen String. Das erste Byte zeigt den Zustand der Shift-, Alternate- und Control-Taste an. Im zweiten Byte wird der Scan-Code der Taste angegeben und erst im vierten Byte steht der ASCII-Code der gedrückten Taste. Das dritte Byte ist unbenutzt.
Nach dem Ersetzen aller INKEY$-Funktionen durch RIGHT$(INKEY$,1) funktionierte das Programm fast ohne Fehler. Fast, denn einige Besonderheiten gilt es zu beachten. Im IDB-Programm werden zum Beispiel Felder gelöscht und in Abhängigkeit des freien Speichers neu dimensioniert. Omikron-Basic kann Felder zwar neu dimensionieren, nur muß die neue Feldgröße insgesamt genausogroß oder größer als das ursprüngliche Feld sein. Löschen kann man ein Stringfeld im Omikron-Basic nicht. Einige weitere Änderungen waren bei den zahlreichen GOTO- und GOSUB-Befehlen erforderlich, da Omikron-Basic nach dem Aufruf einer Subroutine aus einer offenen FOR...NEXT-Schleife die Rücksprungadresse nicht immer fand. Auf diesen Sachverhalt weist der Interpreter hin, sobald das Kommando RUN oder LIST gegeben wird. Durch die EXIT-Funktion konnte aber auch dieses Problem aus der Welt geschafft werden. Mit der EXIT TO-Funktion können Schleifen mit Zielangabe verlassen werden.

Zwei Modi

Beim Omikron-Basic wird zwischen einem Direkt- und einem Edit-Modus unterschieden. Die Eingabe von Programmen kann im Edit-Modus auch ohne Zeilennummern erfolgen. Beim Verlassen des Editors werden Zeilennummern generiert. So kann es passieren, daß man ein Programm ohne Zeilennummern eingibt und beim anschließenden Ausführen eine Fehlermeldung mit Zeilenangabe erhält.
Viele Basic-Programmierer bevorzugen es, bestimmte Programmteile direkt mit der Zeilennummer anzuspringen. Von diesem Programmierstil muß beim Omikron-Basic dringend abgeraten werden, da beim Verlassen des Editors kein Renumber durchgeführt wird. Vielmehr erhält jede Zeile einfach eine Nummer in der Reihenfolge der Eingabe im Editor. Spätestens nach dem Einfügen einiger Zeilen bei abgeschalteter Zeilennummerierung findet das Programm die Zielzeile dann nicht mehr. Hier sollte jeder Programmierer die von Omikron-Basic angebotenen Labels einsetzen.
Der Full-Screen-Editor stellt den gesamten Bildschirmbereich zum Editieren bereit. Die Anzahl von Zeilen ist beim monochromen Monitor in mehreren Auflösungen einstellbar. Beim Aufrufen des Editors stehen 25 Zeilen zu je 80 Zeichen zur Verfügung. Mit einer der acht belegten Funktionstasten kann auf 44 Zeilen zu je 108 Zeichen oder 57 Zeilen zu je 128 Zeichen umgeschaltet werden. Um auch in größeren Programmen nicht den Überblick zu verlieren, kann der Bildschirm in bis zu drei unabhängige Teilbereiche aufgeteilt werden (*Bild 1*). Maximal stehen drei Teilbereiche zu je 57 Zeilen mit 42 Zeichen zur Verfügung. Zwischen den einzelnen Bereichen kann mit dem Cursor hin- und hergesprungen werden.
Alle Funktionen der belegten Funktionstasten und die Sonderfunktionen werden nach einem Druck auf die Help-Taste angezeigt. Eine Klebefolie, die über den Funktionstasten auf den Atari ST geklebt wird, erleichtert die Übersicht. Beim GfA-Basic wird die Belegung immer in den ersten zwei Zeilen im Editor angezeigt. Die Funktionen können hier mit der Maus ausgewählt werden, oder man muß abzählen, der wievielten Funktionstaste welche Funktion zugeordnet ist.
Zwei Funktionstasten können beim Omikron-Basic mit beliebigen Funktionen und Zeichenfolgen doppelt belegt werden. Dazu steht pro Funktionstaste ein bis zu 64 Zeichen großer Speicher zur Verfügung. Die vorbelegten Funktionstasten enthalten Funktionen zum Laden und Speichern von Programmen, zum Definieren, Kopieren, Laden und Speichern von Blöcken, sowie zum Suchen und Ersetzen. Recht nützlich ist die Funktion FIND ALL, die anzeigt, in welchen Programmzeilen eine bestimmte Zeichenfolge vorkommt (*Bild 2*). Damit kann nach Befehlen, Variablen und Zeichenketten gesucht werden. Das Suchen und Ersetzen bestimmter Zeichen läuft auf Wunsch automatisch oder im Einzelschrittmodus ab.
Bei der Eingabe von Programmzeilen erfolgt ein Syntax-Check. Die optische Gestaltung des Programms ist dem Programmierer überlasssen. Automatisches Einrücken von Programmzeilen, wie beim GfA-Basic, gibt es nicht. Ein weiterer Unterschied zum GfA-Basic ist, daß es beim Omikron-Basic kein horizontales Scrollen im Editor gibt. Erreicht eine Programmzeile den rechten Bildschirmrand, wird in der nächsten Bildschirmzeile weitergeschrieben. So werden immer alle Befehle einer Zeile auf dem Bildschirm dargestellt.
Im Omikron-Basic ist der VT52-Standard voll implementiert. Die Funktionen können innerhalb von Basic-Programmen genutzt werden. Dazu schaltet man mit CTRL-A in den ASCII-Modus der Tastatur um, drückt auf die ESC-Taste und gibt danach den Buchstaben für den gewünschten Befehl ein. Die VT52-Befehle können auch im Direkt-Modus eingesetzt werden. Vor dem Start eines Programms prüft der Interpreter, ob alle Schleifen geschlossen sind.

```
13 GOTO Schleife
21 GOTO Schleife
39      GOTO Eingabe1
91 ON Handle GOTO Is_1,Is_2,Is_3,Is_4,Is_5,Is_6,Is_7,Is_8,Is
192 ON Is_Typ(Handle,1) GOTO Is_Search1,Is_Search2
193 Flag=Is_Such2$>Is_Such$: GOTO Is_Search3
195 Flag=Is_Such$(0)<>Is_Such2$: GOTO Is_Search3
```

Bild 2. Hier wurde nach dem Befehl GOTO gesucht

Der Direkt-Modus

Am Direkt-Modus, der zum direkten Ausführen von Befehlen dient, hat mir besonders gefallen, daß man nach dem Listen von Programmen mit dem Cursor wieder bis zum Anfang des Programms steuern kann. Zum Ändern einer Programmzeile muß nicht unbedingt der Editor aufgerufen werden: Das Programm wird mit dem LIST-Kommando

auf den Monitor gebracht, wo mit den Cursortasten dann die zu ändernde Zeile angesteuert wird. Solange mindestens eine Zeile des gelisteten Programms auf dem Bildschirm angezeigt wird, sind alle Programmzeilen erreichbar.
Der Direkt-Modus dient hauptsächlich zum Laden und Speichern von Programmen und zum direkten Ausführen von Programmen. Mit FILES erhält man eine Übersicht über alle Dateien auf dem aktuellen Laufwerk. Vor solche Dateien, die mit der Extension .BAS enden, setzt das Omikron-Basic automatisch den Befehl LOAD. Um dieses Programm zu laden, braucht man nur den Cursor in die entsprechende Zeile zu steuern und Return zu drücken. Auch innerhalb des Editors setzt Omikron nicht auf die Fileselect-Box von GEM. Die Dateibezeichnungen müssen immer eingetippt werden. Damit lassen sich auch Dateien in verschiedenen Unterverzeichnissen ansprechen, was mit der Fileselect-Box nach Meinung von Omikron nicht immer so ohne weiteres funktionieren soll.

Hohe Rechengenauigkeit

Ein besonderer Vorteil von Omikron-Basic gegenüber anderen Basic-Dialekten ist die hohe Genauigkeit, mit der Berechnungen durchgeführt werden. Mit Omikron-Basic kann in einem Zahlenbereich von ±5.11 E±4931 mit 19stelliger Genauigkeit gerechnet werden. Für die Bestimmung der 19ten Stelle ist der Wert der 20sten Stelle ausschlaggebend. Ist die 20ste Stelle ungleich Null, wird die 19te Stelle um den Wert eins erhöht.

Insgesamt kennt das Basic sechs Variablentypen: Neben dem Typ FLAG (0 und 1) noch die Integer-Formate Byte, Word, Long und die Fließkomma-Formate Single und Double. Standardmäßig sind alle Variablen im Omikron-Basic vom Typ long-integer (ergibt einen Zahlenbereich von ±2147483658). Bei der Übernahme von Basic-Programmen aus anderen Dialekten empfiehlt es sich, durch den Befehl DEFSNG „A-Z" in der ersten Programmzeile alle Variablen zum Typ single-float zu deklarieren. Im single-float-Format belegt jede Zahl sechs Byte, zwei Byte für den Exponenten und vier Byte für die Mantisse. Im double-float-Format (Genauigkeit auf 19 Stellen) entfallen auf die Mantisse acht Byte (genau 65 Bit). Bei Berechnungen wird zwischen einfacher und doppelter Genauigkeit unterschieden. Die doppelte Genauigkeit geht natürlich zu Lasten der Arbeitsgeschwindigkeit und des Speicherplatzes. Von den 19 gültigen Stellen werden 17 ausgegeben.

Eine leere FOR...NEXT-Schleife braucht mit 10 000 Durchläufen bei long-integer-Variablen 0,24 Sekunden (GfA-Basic: 0,4 Sekunden). Die gleiche Schleife braucht bei single-float-Variablen 0,54 Sekunden (GfA-Basic: 1,145 Sekunden) und bei double-float-Variablen 0,63 Sekunden. Neben der höheren Genauigkeit (GfA-Basic: 11 Stellen) ist das Omikron-Basic also auch noch schneller. Hinzu kommt der Rechenbereich, der beim GfA-Basic „nur" bis zum Exponenten 154 reicht.
Für Umwandlungen von Zahlen eines Types in einen anderen stehen entsprechende Funktionen bereit. In beiden Basic-Dialekten dürfen String-Variablen bis zu 32767 Zeichen umfassen.
Mit der Genauigkeit von 19 Stellen und dem großen Rechenbereich eignet sich das Omikron-Basic hervorragend für mathematische Anwendungen. Dem trägt das Basic dadurch Rechnung, daß es besondere Funktionen bereithält. Von der Funktion FACT, mit der Fakultäten berechnet werden, bis zu einem Befehlssatz für Matritzenalgebra wird dem Mathematiker einiges geboten. Die zahlreichen Routinen, die man bislang für Aufgaben wie Matritzenmultiplikation, Inversion oder Bestimmung der Determinante einer Matritze schreiben mußte, erledigt das Omikron-Basic mit einem Befehl.
Mit mehreren Funktionen werden Zahlen in ihre interne Speicherdarstellung umgewandelt. Das hat bei der Arbeit mit Dateien, besonders bei solchen mit fixen Datengrößen, Vorteile, da die Zahlen als String bearbeitet werden können. Nach der Umwandlung in die interne Speicherdarstellung ist natürlich auch eine Rückwandlung vom String zur Zahl möglich. Diese Funktionen sollte man nur, wenn man sich mit dem internen Aufbau der Variablen auskennt, mit Vorsicht verwenden.
Bei den Winkelfunktionen bietet Omikron-Basic sehr viel mehr als andere Basic-Dialekte. Gleich 20 Funktionen, vom einfachen Sinus bis zum Areasinus-Hyperbolikus stehen zur Auswahl.
Das Schreiben von umfangreichen Programmen wird mit dem Omikron-Basic zur wahren Freude, da Prozeduren sogar mit lokalen Variablen arbeiten. Prozeduren und Funktionen, die im Edit-Modus eingegeben werden, können auch im Direkt-Modus eingesetzt werden. Editiert man zum Beispiel mit DEF PROC Dollar eine Prozedur zur Umrechnung von Dollarkursen, kann sie vom Direkt-Modus aus mit „Dollar – Parameterübernahme" aufgerufen werden. Funktionen können mehrzeilig und rekursiv sein.

```
Open "O",#1,"testdatei1"
For Test=1 To 10000
  Print #1,Test
Next Test
Close
'
Open "O",#1,"testdatei2"
For Test=1 To 10000
  Print #1,"ABCDEFGHIJK"
Next Test
Close
```

Omikron-Basic mit Diskette	GfA-Basic mit Diskette
Dateitest 1	Dateitest 1
66.04 Sekunden fuer 130001 Byte	204.895 Sekunden fuer 130000 Byte
Dateitest 2	Dateitest 2
38.415 Sekunden fuer 68895 Byte	120.195 Sekunden fuer 58894 Byte
Omikron-Basic mit Festplatte	GfA-Basic mit Festplatte
Dateitest 1	Dateitest 1
19.35 Sekunden fuer 130001 Byte	44.865 Sekunden fuer 130000 Byte
Dateitest 2	Dateitest 2
14.78 Sekunden fuer 68895 Byte	44.105 Sekunden fuer 58894 Byte

Bild 3. Dateien werden mit dem Omikron-Basic schneller auf Datenträger geschrieben als mit dem GfA-Basic

Dateibehandlung

Bei den Befehlen zur Dateibehandlung bietet Omikron-Basic auch etwas mehr

als andere Dialekte. Dateien können von Basic aus kopiert und als Backup gesichert werden. Mit der COMMON-Anweisung wird festgelegt, welche Variablen eines Programms an das mit CHAIN nachgeladene Programm übergeben werden. Wir haben einige sequentielle Dateien mit Omikron- und GfA-Basic geschrieben. Dabei stellten wir fest, daß das Omikron-Basic schneller ist, die erzeugten Dateien in bestimmten Fällen größer sind (*Bild 3*). Das liegt daran, daß der Omikron-Interpreter immer 1024 Byte an das Betriebssystem zum Schreiben auf die Diskette oder Festplatte gibt. Die gestoppten Zeiten beziehen sich auf die Atari-Festplatte SH 204. Wer mit dem Atari ST umfangreiche Dateien verwalten möchte, kommt mit dem Omikron-Basic schneller voran, zumal es Befehle zur indexsequentiellen Dateiverwaltung enthält. Die speziellen ISAM-Befehle sind auf der Diskette als Library mit einem Beispielprogramm enthalten.
Auf der Diskette ist auch die GEM-Library enthalten. Möchte man ein GEM-Programm mit dem Omikron-Basic schreiben, muß an das eigentliche Programm die GEM-Library angehängt werden. Alle VDI- und AES-Funktionen können von Omikron-Basic aus aufgerufen werden. Über die richtige Verwendung dieser Befehle informiert man sich am besten, indem man sich das Mühle-Programm anschaut, das auf der Diskette enthalten ist. Dieses Programm installiert ein leistungsstarkes Mühlespiel als Desk-Accessorie.

Multitasking

Das Omikron-Basic läßt einen interruptgesteuerten Aufruf von Programmteilen zu. In Abhängigkeit von der Systemuhr kann man mit ON TIMER GOSUB in frei bestimmbaren Abständen Prozeduren aufrufen. Damit läßt sich so etwas wie Multitasking simulieren. Mit dem Befehl ON TIMER 1 GOSUB -UHR erreicht man zum Beispiel, daß der Interpreter nach dem Ablauf einer Sekunde die Prozedur „Uhr" anspringt. In dieser Prozedur braucht nur die Anweisung PRINT @(0,0) TIME$ zu stehen, und schon zeigt das Programm immer die aktuelle Zeit an. Einige Befehle – zum Beispiel der SORT-Befehl – lassen keinen Interrupt-Aufruf zu.
Außer vom Timer kann man Prozeduren noch durch Tasten (Maustaste, Help-Taste, oder beliebige andere Taste) aufrufen. In dem Programm aus *Bild 4* laufen vier Programme gleichzeitig ab. Zum einen die leere FOR...NEXT-Schleife und zum anderen drei Interrupt-Aufrufe.

```
0 ON TIMER 1 GOSUB Uhr
1 ON HELP GOSUB Hilfe
2 ON MOUSEBUT GOSUB Maus
3
4-Schleife
5 FOR T=1 TO 100000
6 'Hier steht Ihr Programm
7 NEXT T
8 GOTO Schleife
9-Hilfe
10 PRINT "HILFE", TIME$ : RETURN
11-Uhr: PRINT @(0,0); TIME$
12 RETURN
13-Maus
14 PRINT "Maus"; TIME$ : RETURN
```

Bild 4. Multitasking ist mit dem Omikron-Basic kein Problem

Selbst wenn alle drei Bedingungen erfüllt sind (Maustaste und Help-Taste gedrückt und Timer-Interrupt vorhanden), läuft das Hauptprogramm weiter. Aus einer Prozedur heraus springt das Programm an die Stelle im Hauptprogramm zurück, an der der Interrupt ausgelöst wurde. Ausflüge in die Assemblerprogrammierung sind mit Omikron-Basic nicht mehr erforderlich, möchte man mehrere Prozesse gleichzeitig ablaufen lassen. Wer es trotzdem nicht lassen kann, von Basic-Programmen aus Routinen in Maschinensprache aufzurufen, wird sich darüber freuen, daß sich Omikron-Basic nach einem Reset wieder meldet. Auch das gerade im Speicher befindliche Programm geht dabei nicht verloren.

Einige Besonderheiten

Von Omikron-Basic-Programmen aus können Fileselect-Boxen zur Auswahl von Dateien dargestellt werden. Mit zwei Hardcopy-Befehlen können Bildschirminhalte als Grafik oder Text ausgegeben werden. Zum Sortieren von Feldern stellt das Basic den Befehl SORT zur Verfügung, der nach dem Quicksort-Verfahren arbeitet (*Bild 5*). Zum Sortieren von 1000 Elementen benötigt das Omikron-Basic etwa 1,7 Sekunden. Eine mit GfA-Basic geschriebene Quicksort-Routine leistet das gleiche in etwa 7 Sekunden. Bei 10 000 Elementen wird der Unterschied noch deutlicher: Omikron in 24 Sekunden, GfA in fast 90 Sekunden. Ohne viel Aufwand – mit nur einem Befehl – sortiert Omikron-Basic Zeichenfolgen in eindimensionalen Feldern sogar mit Umlauten richtig. Ein zweites Feld kann in gleicher Reihenfolge mitsortiert werden. Der SORT-Befehl ist auf Zeichen- und Zahlenfelder anwendbar. Für Programmierer, die sich mit der Grafik des Atari ST beschäftigen wollen, ist der Befehl CLIP wichtig, mit dem das Clipping Rectangle – der Zeichenbereich – festgelegt wird. Drei Bildschirme können unabhängig voneinander programmiert und manipuliert werden. Zusätzlich zu den üblichen Grafik-Befehlen kennt das Omikron-Basic den Befehl BITBLT, mit dem man Ausschnitte aus dem Bildschirm an andere Bildschirmpositionen oder in den Speicher kopieren kann. In den Speicher abgelegte Bildschirmausschnitte sind natürlich auch wieder darstellbar. Dem Grafiker öffnen sich hiermit sehr interessante Effekte. Für Abwechslung sorgen auch Sprite-Befehle, mit denen acht Sprites definiert und bewegt werden.

```
DIM A$(X)
FOR I=1 TO X
FOR T=1 TO 10
Z$=Z$+ CHR$(65+ RND(26))
NEXT T
A$(I)=Z$:Z$=""
NEXT I
SORT A$(0)
' Für     X        Sekunden
'        100          0,105
'        500          0,72
'       1000          1,67
'       5000          10,9
'      10000          24,4
```

Bild 5. Mit dem SORT-Befehl werden große Zahlen- und Zeichenfelder sehr schnell sortiert

Alle Variablen, die in einem Programm benutzt werden, lassen sich mit DUMP mit ihren Inhalten anzeigen. Das funktioniert auch sofort nach dem Laden eines Programms in den Speicher. Tastatureingaben werden mit der Anweisung INPUT USING auf richtige Eingaben kontrolliert. Damit können umfangreiche Tastaturabfrage-Routinen entfallen.
Der Omikron-Basic-Interpreter ist sehr schnell und arbeitet genauer als zum Beispiel das GfA-Basic, dessen Interpreter den Vorteil hat, daß er nur sauber strukturierte Programme erzeugen kann. Spaghetti-Code, wie er in vielen Basic-Programmen immer wieder entsteht, kann mit dem Omikron-Basic hervorragend produziert werden. Die einzige Beschränkung, die wir am Omikron-Interpreter feststellen konnten, liegt bei den maximal 32767 Zeilen Programmcode. Damit kann sicher jedes Basic-Programm – auch auf dem Atari ST – zum Laufen gebracht werden.

ZENITH Z 148
Der IBM-kompatible Profi-Computer

PROZESSOR:	8088 B (Taktfrequenz 8 MHz umschaltbar auf 4,77 MHz)
HAUPTSPEICHER:	512 K RAM (erweiterbar auf 640 kB)
LAUFWERKE:	2 Diskettenlaufwerke je 360 kB
SCHNITTSTELLEN:	Druckeranschluß Centronics (parallel) und V-24 (seriell)
BETRIEBSSYSTEME:	MS-DOS 3.1, GW-BASIC, Rel. 2.0 (mit deutschen Handbüchern)
SONSTIGE AUSSTATTUNGEN:	Videografik (Auflösung 640 x 200 Punkte), Tastatur nach DIN, incl. hochauflösendem Video-Monitor

Auch mit 30 Megabyte-Festplatte lieferbar!

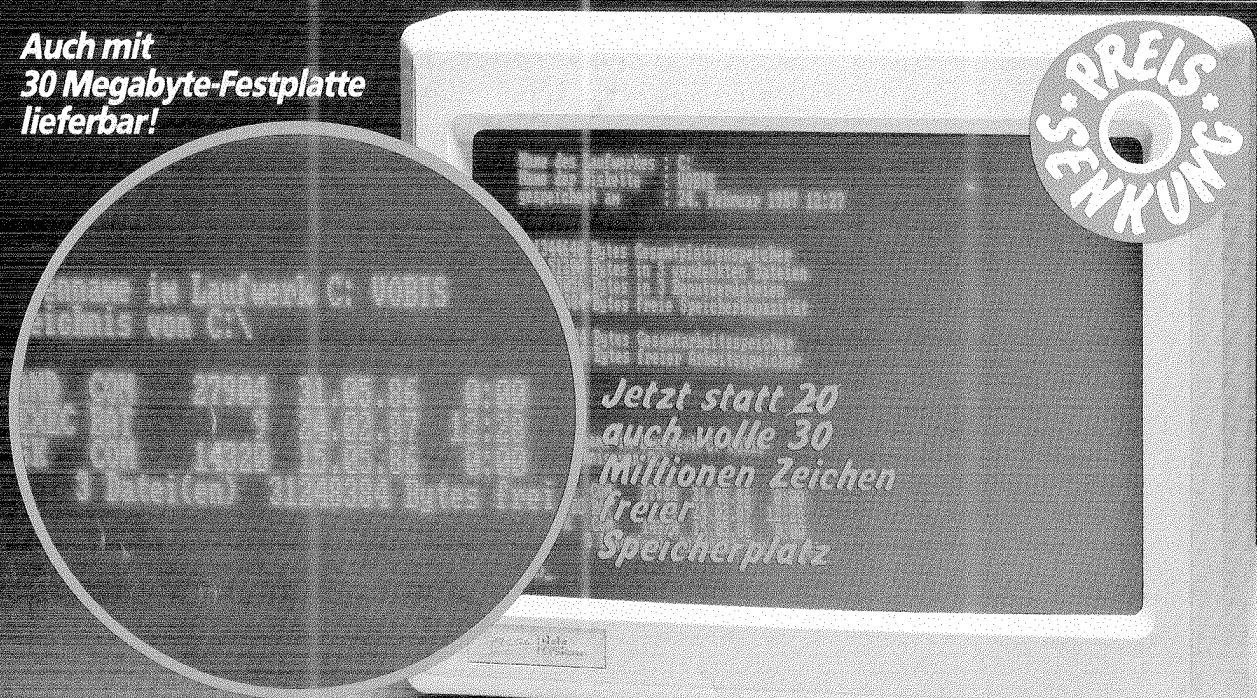
PREIS-SENKUNG

Z148 (wie oben beschrieben) nur noch **1995.-**

Mit **20 MB-**Festspeicherplatte nur noch **2875.-**

Mit **30 MB-**Festspeicherplatte nur noch **3345.-**

30 MB-Festspeicherplatte incl. Controller einzeln **1395.-**



ZENITH Z148
Ein bißchen kompatibler!
Denn die Taktfrequenz des 8088 Microcomputers ist umschaltbar: Von schneller 8 Mhz auf 4,77 Mhz wie beim Original IBM-PC.
Dadurch ist der
ZENITH Z148
kompatibler als viele
Nur 8 Mhz
sonst!

© pSE-DESIGN

kompetent + preiswert

WICHTIG: Herstellerbedingte Lieferzeiten! Aufgrund erhöhter Nachfrage ist nicht immer alles sofort lieferbar!

VOBIS
Deutschlands umsatzgrößter Microcomputer-Spezialist

VERSAND-ZENTRALE:
Postfach 1778
Rotter Bruch 32/34
5100 AACHEN
☎ 0241/50 00 81
Tx 832389 vobis d

FILIALEN:

AACHEN
Viktoriastr. 74 · 0241/54 31 00

AACHEN
Pontstraße 60 · 0241/3 38 06

BERLIN 30
Kurfürstenstr. 101 · 030/2 13 94 80

BIELEFELD
Herforder Str. 106 · 0521/6 38 78

BREMEN
Violenstraße 37 · 0421/32 04 20

DORTMUND
Hamburger Str. 110 · 0231/57 30 72

DÜSSELDORF
Wielandstr. 21 · 0211/35 99 64

FRANKFURT
Frankenallee 207/209 · 069/73 40 49

HAMBURG
Krohnskamp 15 · 040/2 79 46 76

HANNOVER
Berliner Allee 47 · 0511/81 65 71

KARLSRUHE
Kriegsstr. 27/29 (am BGH) 0721/37 82 68

KONSTANZ
Kreuzlinger Str. 18 · 07531/1 55 60

KÖLN
Mathiasstr. 24-26 · 0221/24 86 42

MÜNCHEN
Aberlestraße 3 · 089/77 21 10

NÜRNBERG
Vordere Ledergasse 8 · 0911/23 29 95

STUTTGART
Marienstr. 11-13 · 0711/60 63 36